Chap. VI : Les processus

Laurent Poinsot

UMR 7030 - Université Paris 13 - Institut Galilée

Cours “Architecture et Système”

Un processus est un programme en cours d'exécution. Il faut d'emblée faire la différence entre un programme qui est un fichier inerte regroupant des instructions pour le CPU et un processus qui un élément actif (on parle donc de " programme en cours d'exécution " pour un processus).

Un processus est un programme en cours d'exécution. Il faut d'emblée faire la différence entre un programme qui est un fichier inerte regroupant des instructions pour le CPU et un processus qui un élément actif (on parle donc de " programme en cours d'exécution " pour un processus).

Un processus est un programme en cours d'exécution. Il faut d'emblée faire la différence entre un programme qui est un fichier inerte regroupant des instructions pour le CPU et un processus qui un élément actif (on parle donc de " programme en cours d'exécution " pour un processus).

Figeons un processus pour en observer ses composantes. Nous trouvons :

- des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus ;
- la valeur des registres du processeur lors de l'exécution ;
- les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le contexte d'exécution d'un processus ou plus simplement le contexte.

Figeons un processus pour en observer ses composantes. Nous trouvons :

- des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus ;
- la valeur des registres du processeur lors de l'exécution ;
- les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le contexte d'exécution d'un processus ou plus simplement le contexte.

Figeons un processus pour en observer ses composantes. Nous trouvons :

- des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus ;
- la valeur des registres du processeur lors de l'exécution ;
- les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le contexte d'exécution d'un processus ou plus simplement le contexte.

Figeons un processus pour en observer ses composantes. Nous trouvons :

- des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus ;
- la valeur des registres du processeur lors de l'exécution ;
- les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le contexte d'exécution d'un processus ou plus simplement le contexte.

Figeons un processus pour en observer ses composantes. Nous trouvons :

- des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus ;
- la valeur des registres du processeur lors de l'exécution ;
- les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le contexte d'exécution d'un processus ou plus simplement le contexte.

Figeons un processus pour en observer ses composantes. Nous trouvons :

- des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus ;
- la valeur des registres du processeur lors de l'exécution ;
- les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le contexte d'exécution d'un processus ou plus simplement le contexte.

Un processus n'est pas continuellement en train de s'exécuter. Si la machine comporte n processeurs identiques, à un instant donné il y a au maximum n processus actifs (c'est-à-dire en cours d'exécution). En fait, parmi tous les processus qui sont susceptibles de s'exécuter, seulement un petit nombre s'exécutent réellement. L'allocation du processeur aux processus qui le réclament est appelée l'ordonnancement du processeur. On en reparlera dans la suite du cours.

Un processus n'est pas continuellement en train de s'exécuter. Si la machine comporte n processeurs identiques, à un instant donné il y a au maximum n processus actifs (c'est-à-dire en cours d'exécution).

En fait, parmi tous les processus qui sont susceptibles de s'exécuter, seulement un petit nombre s'exécutent réellement. L'allocation du processeur aux processus qui le réclament est appelée l'ordonnancement du processeur. On en reparlera dans la suite du cours.

Un processus n'est pas continuellement en train de s'exécuter. Si la machine comporte n processeurs identiques, à un instant donné il y a au maximum n processus actifs (c'est-à-dire en cours d'exécution). En fait, parmi tous les processus qui sont susceptibles de s'exécuter, seulement un petit nombre s'exécutent réellement. L'allocation du processeur aux processus qui le réclament est appelée l'ordonnancement du processeur. On en reparlera dans la suite du cours.

Un processus n'est pas continuellement en train de s'exécuter. Si la machine comporte n processeurs identiques, à un instant donné il y a au maximum n processus actifs (c'est-à-dire en cours d'exécution). En fait, parmi tous les processus qui sont susceptibles de s'exécuter, seulement un petit nombre s'exécutent réellement. L'allocation du processeur aux processus qui le réclament est appelée l'ordonnancement du processeur. On en reparlera dans la suite du cours.

Un processus n'est pas continuellement en train de s'exécuter. Si la machine comporte n processeurs identiques, à un instant donné il y a au maximum n processus actifs (c'est-à-dire en cours d'exécution). En fait, parmi tous les processus qui sont susceptibles de s'exécuter, seulement un petit nombre s'exécutent réellement. L'allocation du processeur aux processus qui le réclament est appelée l'ordonnancement du processeur. On en reparlera dans la suite du cours.

L'état opérationnel d'un processus est un moyen de représenter les différentes étapes de ce processus telles qu'elles sont gérées par le système d'exploitation. Le schéma suivant montre les divers états dans lesquels, dans une première approche intuitive, peut se trouver un processus ("`états d'un processus.pdf`").

L'état opérationnel d'un processus est un moyen de représenter les différentes étapes de ce processus telles qu'elles sont gérées par le système d'exploitation. Le schéma suivant montre les divers états dans lesquels, dans une première approche intuitive, peut se trouver un processus ("`états d'un processus.pdf`").

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

- Initialement, un processus est connu du système mais son exécution n'a pas débuté.
- Lorsqu'il est initialisé, il devient prêt à être exécuté (1).
- Lors de l'allocation du processeur à ce processus il devient actif (2). Trois cas peuvent alors se présenter :
  - Le processus se termine (3).
  - Le processus est en attente (5) d'un événement et dès sa réception il redeviendra prêt (6).
  - Le processus est suspendu et se remet dans l'état prêt (4). Il y a réquisition ou préemption du processeur. Dans ce cas, l'O.S. enlève le processeur au processus qui la détient. Ce mécanisme sera vu en détail au chapitre traitant de l'allocation du processeur.

La notion " d'attente d'un événement " mérite par son importance et sa complexité un petit exemple. Un éditeur de texte enchaîne continuellement la boucle suivante :

```
répéter
    lire un caractère
    traiter ce caractère
jusqu'à ...
```

Lorsque le processus éditeur est actif il adresse une requête à l'O.S. pour lui demander une opération d'E/S (la lecture d'un caractère).

La notion " d'attente d'un événement " mérite par son importance et sa complexité un petit exemple. Un éditeur de texte enchaîne continuellement la boucle suivante :

```
répéter
    lire un caractère
    traiter ce caractère
jusqu'à ...
```

Lorsque le processus éditeur est actif il adresse une requête à l'O.S. pour lui demander une opération d'E/S (la lecture d'un caractère).

La notion " d'attente d'un événement " mérite par son importance et sa complexité un petit exemple. Un éditeur de texte enchaîne continuellement la boucle suivante :

```
répéter
    lire un caractère
    traiter ce caractère
jusqu'à ...
```

Lorsque le processus éditeur est actif il adresse une requête à l'O.S. pour lui demander une opération d'E/S (la lecture d'un caractère).

La notion " d'attente d'un événement " mérite par son importance et sa complexité un petit exemple. Un éditeur de texte enchaîne continuellement la boucle suivante :

```
répéter
    lire un caractère
    traiter ce caractère
jusqu'à ...
```

Lorsque le processus éditeur est actif il adresse une requête à l'O.S. pour lui demander une opération d'E/S (la lecture d'un caractère).

Deux cas se présentent :

- s'il existe un caractère dans le tampon d'entrée (c'est-à-dire qui vient d'être taper au clavier), alors ce dernier est renvoyé par l'O.S. ;
- si le tampon d'entrée est vide (c'est-à-dire qu'aucun caractère ne vient d'être tapé), alors l'O.S. va " endormir " le processus en changeant son état. Lorsque l'utilisateur frappe une touche du clavier, l'O.S. (qui avait préalablement sauvegardé la demande de l'éditeur en stockant le caractère tapé au clavier dans le tampon) " réveille " l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Deux cas se présentent :

- s'il existe un caractère dans le tampon d'entrée (c'est-à-dire qui vient d'être taper au clavier), alors ce dernier est renvoyé par l'O.S. ;
- si le tampon d'entrée est vide (c'est-à-dire qu'aucun caractère ne vient d'être tapé), alors l'O.S. va " endormir " le processus en changeant son état. Lorsque l'utilisateur frappe une touche du clavier, l'O.S. (qui avait préalablement sauvegardé la demande de l'éditeur en stockant le caractère tapé au clavier dans le tampon) " réveille " l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Deux cas se présentent :

- s'il existe un caractère dans le tampon d'entrée (c'est-à-dire qui vient d'être taper au clavier), alors ce dernier est renvoyé par l'O.S. ;
- si le tampon d'entrée est vide (c'est-à-dire qu'aucun caractère ne vient d'être tapé), alors l'O.S. va " endormir " le processus en changeant son état. Lorsque l'utilisateur frappe une touche du clavier, l'O.S. (qui avait préalablement sauvegardé la demande de l'éditeur en stockant le caractère tapé au clavier dans le tampon) " réveille " l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Deux cas se présentent :

- s'il existe un caractère dans le tampon d'entrée (c'est-à-dire qui vient d'être taper au clavier), alors ce dernier est renvoyé par l'O.S. ;
- si le tampon d'entrée est vide (c'est-à-dire qu'aucun caractère ne vient d'être tapé), alors l'O.S. va " endormir " le processus en changeant son état. Lorsque l'utilisateur frappe une touche du clavier, l'O.S. (qui avait préalablement sauvegardé la demande de l'éditeur en stockant le caractère tapé au clavier dans le tampon) " réveille " l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Deux cas se présentent :

- s'il existe un caractère dans le tampon d'entrée (c'est-à-dire qui vient d'être taper au clavier), alors ce dernier est renvoyé par l'O.S. ;
- si le tampon d'entrée est vide (c'est-à-dire qu'aucun caractère ne vient d'être tapé), alors l'O.S. va " endormir " le processus en changeant son état. Lorsque l'utilisateur frappe une touche du clavier, l'O.S. (qui avait préalablement sauvegardé la demande de l'éditeur en stockant le caractère tapé au clavier dans le tampon) " réveille " l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Deux cas se présentent :

- s'il existe un caractère dans le tampon d'entrée (c'est-à-dire qui vient d'être taper au clavier), alors ce dernier est renvoyé par l'O.S. ;
- si le tampon d'entrée est vide (c'est-à-dire qu'aucun caractère ne vient d'être tapé), alors l'O.S. va " endormir " le processus en changeant son état. Lorsque l'utilisateur frappe une touche du clavier, l'O.S. (qui avait préalablement sauvegardé la demande de l'éditeur en stockant le caractère tapé au clavier dans le tampon) " réveille " l'éditeur qui pourra ainsi devenir actif et traiter ce fameux caractère.

Un processus est caractérisé dans le système par :

- un identificateur ou numéro (par exemple le PID, pour Process IDentification, dans le système UNIX) ;
- un état opérationnel (l'un des cinq vus précédemment) ;
- un contexte ;
- des informations comme les priorités, la date de démarrage, la filiation ;
- des statistiques calculées par l'O.S. telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Un processus est caractérisé dans le système par :

- un identificateur ou numéro (par exemple le PID, pour Process IDentification, dans le système UNIX) ;
- un état opérationnel (l'un des cinq vus précédemment) ;
- un contexte ;
- des informations comme les priorités, la date de démarrage, la filiation ;
- des statistiques calculées par l'O.S. telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Un processus est caractérisé dans le système par :

- un identificateur ou numéro (par exemple le PID, pour Process IDentification, dans le système UNIX) ;
- un état opérationnel (l'un des cinq vus précédemment) ;
- un contexte ;
- des informations comme les priorités, la date de démarrage, la filiation ;
- des statistiques calculées par l'O.S. telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Un processus est caractérisé dans le système par :

- un identificateur ou numéro (par exemple le PID, pour Process IDentification, dans le système UNIX) ;
- un état opérationnel (l'un des cinq vus précédemment) ;
- un contexte ;
- des informations comme les priorités, la date de démarrage, la filiation ;
- des statistiques calculées par l'O.S. telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Un processus est caractérisé dans le système par :

- un identificateur ou numéro (par exemple le PID, pour Process IDentification, dans le système UNIX) ;
- un état opérationnel (l'un des cinq vus précédemment) ;
- un contexte ;
- des informations comme les priorités, la date de démarrage, la filiation ;
- des statistiques calculées par l'O.S. telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Un processus est caractérisé dans le système par :

- un identificateur ou numéro (par exemple le PID, pour Process IDentification, dans le système UNIX) ;
- un état opérationnel (l'un des cinq vus précédemment) ;
- un contexte ;
- des informations comme les priorités, la date de démarrage, la filiation ;
- des statistiques calculées par l'O.S. telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus. Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus. Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus. Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus. Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus. Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus.

Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Ces informations sont regroupées dans un bloc de contrôle de processus ou PCB (Process Control Block). Le système maintient donc un PCB pour chaque processus reconnu. Ce PCB est une mine d'informations. Lorsqu'un processus quitte l'état actif, son PCB est mis à jour et la valeur des registres du processeur y est sauvegardé. Pour que ce même processus redevienne actif, l'O.S. recharge les registres du processeur à partir des valeurs sauvegardées dans le PCB, il change l'état et finalement il redémarre l'exécution du processus. Nous verrons plus tard dans quelles conditions un processus peut " perdre " le processeur.

Hormis les processus, le système maintient également un certain nombre de files d'attente qui regroupent les blocs de contrôle des processus. On trouve ainsi,

- la file des processus prêts,
- la file des processus connus,
- la file des processus en attente.

La figure suivante illustre le chemin suivi par notre éditeur de texte dans les files du système lors de sa demande d'un caractère sur l'unité d'entrée standard ("`Processus et files.pdf`").

Hormis les processus, le système maintient également un certain nombre de files d'attente qui regroupent les blocs de contrôle des processus. On trouve ainsi,

- la file des processus prêts,
- la file des processus connus,
- la file des processus en attente.

La figure suivante illustre le chemin suivi par notre éditeur de texte dans les files du système lors de sa demande d'un caractère sur l'unité d'entrée standard ("`Processus et files.pdf`").

Hormis les processus, le système maintient également un certain nombre de files d'attente qui regroupent les blocs de contrôle des processus. On trouve ainsi,

- la file des processus prêts,
- la file des processus connus,
- la file des processus en attente.

La figure suivante illustre le chemin suivi par notre éditeur de texte dans les files du système lors de sa demande d'un caractère sur l'unité d'entrée standard ("`Processus et files.pdf`").

Hormis les processus, le système maintient également un certain nombre de files d'attente qui regroupent les blocs de contrôle des processus. On trouve ainsi,

- la file des processus prêts,
- la file des processus connus,
- la file des processus en attente.

La figure suivante illustre le chemin suivi par notre éditeur de texte dans les files du système lors de sa demande d'un caractère sur l'unité d'entrée standard ("`Processus et files.pdf`").

Hormis les processus, le système maintient également un certain nombre de files d'attente qui regroupent les blocs de contrôle des processus. On trouve ainsi,

- la file des processus prêts,
- la file des processus connus,
- la file des processus en attente.

La figure suivante illustre le chemin suivi par notre éditeur de texte dans les files du système lors de sa demande d'un caractère sur l'unité d'entrée standard ("`Processus et files.pdf`").

Hormis les processus, le système maintient également un certain nombre de files d'attente qui regroupent les blocs de contrôle des processus. On trouve ainsi,

- la file des processus prêts,
- la file des processus connus,
- la file des processus en attente.

La figure suivante illustre le chemin suivi par notre éditeur de texte dans les files du système lors de sa demande d'un caractère sur l'unité d'entrée standard ("`Processus et files.pdf`").

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus sont les principaux éléments actifs du système. Dans ce cadre, il est logique que la création d'un nouveau processus puisse être demandée par un processus. Il existe donc une filiation entre processus père et le(s) processus fils. Lors du démarrage de la machine, l'O.S. lance un processus qui est orphelin puisqu'il n'a pas de père. Ce processus est souvent appelé `init`. Le premier rôle de ce processus est de lancer des fils qui auront chacun une fonction dans l'organisation générale de la machine. Par exemple,

- un processus pour gérer les E/S asynchrones avec les terminaux,
- un processus pour gérer les connexions au système avec demande et vérification d'un nom d'utilisateur et d'un mot de passe,
- un processus pour gérer l'allocation de la CPU (processeur) aux processus !

Les processus fils de `init`, et `init`, lui-même font partie du système d'exploitation. Ils s'exécutent donc avec des droits étendus. Nous les appellerons les processus système ou démons (daemons) par opposition aux processus utilisateur. Le système d'exploitation est donc composé d'un noyau résident qui ne s'exécute que sur demande explicite (interruptions) et d'un ensemble de processus système qui ont chacun une fonction précise à assurer.

Les processus fils de `init`, et `init`, lui-même font partie du système d'exploitation. Ils s'exécutent donc avec des droits étendus.

Nous les appellerons les processus système ou démons (daemons) par opposition aux processus utilisateur. Le système d'exploitation est donc composé d'un noyau résident qui ne s'exécute que sur demande explicite (interruptions) et d'un ensemble de processus système qui ont chacun une fonction précise à assurer.

Les processus fils de `init`, et `init`, lui-même font partie du système d'exploitation. Ils s'exécutent donc avec des droits étendus. Nous les appellerons les processus système ou démons (daemons) par opposition aux processus utilisateur. Le système d'exploitation est donc composé d'un noyau résident qui ne s'exécute que sur demande explicite (interruptions) et d'un ensemble de processus système qui ont chacun une fonction précise à assurer.

Les processus fils de `init`, et `init`, lui-même font partie du système d'exploitation. Ils s'exécutent donc avec des droits étendus. Nous les appellerons les processus système ou démons (daemons) par opposition aux processus utilisateur. Le système d'exploitation est donc composé d'un noyau résident qui ne s'exécute que sur demande explicite (interruptions) et d'un ensemble de processus système qui ont chacun une fonction précise à assurer.

Les processus fils de `init`, et `init`, lui-même font partie du système d'exploitation. Ils s'exécutent donc avec des droits étendus. Nous les appellerons les processus système ou démons (daemons) par opposition aux processus utilisateur. Le système d'exploitation est donc composé d'un noyau résident qui ne s'exécute que sur demande explicite (interruptions) et d'un ensemble de processus système qui ont chacun une fonction précise à assurer.

Après avoir étudié les processus au sein des systèmes d'exploitation quelconques, on s'intéresse maintenant à leurs incarnations dans les O.S. de la famille Unix (GNU/Linux, SUN-Solaris, Minix, XINU, Mac OS S, HP-UX, Windows SFU " Services For Unix "). La programmation système se fera en effet en langage C sous Linux. Il nous faut donc connaître la façon de les manipuler à l'aide du langage C.

Après avoir étudié les processus au sein des systèmes d'exploitation quelconques, on s'intéresse maintenant à leurs incarnations dans les O.S. de la famille Unix (GNU/Linux, SUN-Solaris, Minix, XINU, Mac OS S, HP-UX, Windows SFU " Services For Unix "). La programmation système se fera en effet en langage C sous Linux. Il nous faut donc connaître la façon de les manipuler à l'aide du langage C.

Après avoir étudié les processus au sein des systèmes d'exploitation quelconques, on s'intéresse maintenant à leurs incarnations dans les O.S. de la famille Unix (GNU/Linux, SUN-Solaris, Minix, XINU, Mac OS S, HP-UX, Windows SFU " Services For Unix "). La programmation système se fera en effet en langage C sous Linux. Il nous faut donc connaître la façon de les manipuler à l'aide du langage C.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Un processus est un ensemble d'octets (écrits en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. Un processus UNIX se décompose en :

1. un espace d'adressage (visible par l'utilisateur/programmeur) ;
2. Le bloc de contrôle du processus (PCB en anglais) lui-même décomposé en :
   - une entrée dans la table des processus du noyau `struct proc` définie dans la librairie `<sys/proc.h>`.
   - une structure `struct user` appelée zone u définie dans `<sys/user.h>`.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus. Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus. Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus.
- Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus. Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus. Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus.

Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus. Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Les processus sous Unix apportent :

- La multiplicité des exécutions. En effet, plusieurs processus peuvent être l'exécution d'un même programme.
- La protection des exécutions. Un processus ne peut exécuter que ses instructions propres et ce de façon séquentielle ; il ne peut pas exécuter des instructions appartenant à un autre processus. Les processus sous UNIX communiquent entre eux et avec le reste du monde grâce aux appels système.

C'est ce système de communication entre processus sous Unix qui constitue le fil conducteur de ce cours de programmation système.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

Création d'un processus - `fork()`

Sous UNIX la création de processus est réalisée en langage C par l'appel système :

```
pid_t fork(void);
```

Tous les processus - sauf le processus d'identification 0 (le `main`), sont créés par un appel système à `fork`. Le processus qui appelle le `fork` est appelé processus père. Le nouveau processus créé par un `fork` est appelé processus fils. Tout processus a un seul processus père. Tout processus peut avoir zéro ou plusieurs processus fils.

PID

Chaque processus est identifié de façon unique par un numéro : son PID (Process IDentification). Le processus de PID=0 est créé au démarrage de la machine. Ce processus a toujours un rôle spécial pour le système (surtout pour la gestion de la mémoire). Le processus zéro crée, grâce à un appel de `fork`, le processus `init` dont le PID est égal à 1. Ce processus de PID=1 est l'ancêtre de tous les autres processus (le processus 0 ne réalisant plus de`fork()`) et c'est lui qui accueille tous les processus orphelins de père (ceci afin de collecter les information à la mort de chaque processus).

PID

Chaque processus est identifié de façon unique par un numéro : son PID (Process IDentification). Le processus de PID=0 est créé au démarrage de la machine. Ce processus a toujours un rôle spécial pour le système (surtout pour la gestion de la mémoire). Le processus zéro crée, grâce à un appel de `fork`, le processus `init` dont le PID est égal à 1. Ce processus de PID=1 est l'ancêtre de tous les autres processus (le processus 0 ne réalisant plus de`fork()`) et c'est lui qui accueille tous les processus orphelins de père (ceci afin de collecter les information à la mort de chaque processus).

PID

Chaque processus est identifié de façon unique par un numéro : son PID (Process IDentification). Le processus de PID=0 est créé au démarrage de la machine. Ce processus a toujours un rôle spécial pour le système (surtout pour la gestion de la mémoire). Le processus zéro crée, grâce à un appel de `fork`, le processus `init` dont le PID est égal à 1. Ce processus de PID=1 est l'ancêtre de tous les autres processus (le processus 0 ne réalisant plus de `fork()`) et c'est lui qui accueille tous les processus orphelins de père (ceci afin de collecter les information à la mort de chaque processus).

PID

Chaque processus est identifié de façon unique par un numéro : son PID (Process IDentification). Le processus de PID=0 est créé au démarrage de la machine. Ce processus a toujours un rôle spécial pour le système (surtout pour la gestion de la mémoire). Le processus zéro crée, grâce à un appel de `fork`, le processus `init` dont le PID est égal à 1. Ce processus de PID=1 est l'ancêtre de tous les autres processus (le processus 0 ne réalisant plus de `fork()`) et c'est lui qui accueille tous les processus orphelins de père (ceci afin de collecter les information à la mort de chaque processus).

PID

Chaque processus est identifié de façon unique par un numéro : son PID (Process IDentification). Le processus de PID=0 est créé au démarrage de la machine. Ce processus a toujours un rôle spécial pour le système (surtout pour la gestion de la mémoire). Le processus zéro crée, grâce à un appel de `fork`, le processus `init` dont le PID est égal à 1. Ce processus de PID=1 est l'ancêtre de tous les autres processus (le processus 0 ne réalisant plus de`fork()`) et c'est lui qui accueille tous les processus orphelins de père (ceci afin de collecter les information à la mort de chaque processus).

PID

Chaque processus est identifié de façon unique par un numéro : son PID (Process IDentification). Le processus de PID=0 est créé au démarrage de la machine. Ce processus a toujours un rôle spécial pour le système (surtout pour la gestion de la mémoire). Le processus zéro crée, grâce à un appel de `fork`, le processus `init` dont le PID est égal à 1. Ce processus de PID=1 est l'ancêtre de tous les autres processus (le processus 0 ne réalisant plus de`fork()`) et c'est lui qui accueille tous les processus orphelins de père (ceci afin de collecter les information à la mort de chaque processus).

Les compilateurs nous permettent de créer des fichiers exécutables.

Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

Les compilateurs nous permettent de créer des fichiers exécutables. Ces fichiers ont le format suivant qui permet au noyau de les transformer en processus :

- Une en-tête qui décrit l'ensemble du fichier.
- La taille à allouer pour les variables non initialisées.
- Une section `TEXT` qui contient le code (en langage machine).
- Une section de données (`DATA`) codée en langage machine qui contient les données initialisées.
- Éventuellement d'autres sections : table des symboles pour le débogeur, Images, ICONS, etc.

Pour plus d'informations se reporter au manuel `a.out.h` sur la machine.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

L'appel système `execve` change l'exécutable du processus courant en chargeant un nouvel exécutable. Les régions associée au processus sont préalablement libérées :

```
int execve(/* plusieurs formats */);
```

Pour chaque section de l'exécutable, une région en mémoire est allouée. Soit au moins les régions :

- le code ;
- les données initialisées ;

Mais aussi les régions :

- de la pile ;
- du tas.

La région de la pile : c'est une pile de structures qui sont empilées et dépilées lors de l'appel ou le retour de fonction. Le pointeur de pile, un des registres de l'unité centrale, indique la profondeur courante de la pile. Le tas est une zone où est réalisée l'allocation dynamique avec les fonctions du type `Xalloc()`.

La région de la pile : c'est une pile de structures qui sont empilées et dépilées lors de l'appel ou le retour de fonction. Le pointeur de pile, un des registres de l'unité centrale, indique la profondeur courante de la pile. Le tas est une zone où est réalisée l'allocation dynamique avec les fonctions du type `Xalloc()`.

La région de la pile : c'est une pile de structures qui sont empilées et dépilées lors de l'appel ou le retour de fonction. Le pointeur de pile, un des registres de l'unité centrale, indique la profondeur courante de la pile. Le tas est une zone où est réalisée l'allocation dynamique avec les fonctions du type `Xalloc()`.

La région de la pile : c'est une pile de structures qui sont empilées et dépilées lors de l'appel ou le retour de fonction. Le pointeur de pile, un des registres de l'unité centrale, indique la profondeur courante de la pile. Le tas est une zone où est réalisée l'allocation dynamique avec les fonctions du type `Xalloc()`.

Tous les processus sont associés à une entrée dans la table des processus qui est interne au noyau. De plus, le noyau alloue pour chaque processus une structure appelée zone u , qui contient des données privées du processus, uniquement manipulables par le noyau. La table des processus nous permet d'accéder à la table des régions. Les structures de régions de la table des régions contiennent des informations sur le type, les droits d'accès et la localisation (adresses en mémoire ou adresses sur disque) de la région. Seule la zone u du processus courant est manipulable par le noyau, les autres sont inaccessibles.

Tous les processus sont associés à une entrée dans la table des processus qui est interne au noyau. De plus, le noyau alloue pour chaque processus une structure appelée zone u , qui contient des données privées du processus, uniquement manipulables par le noyau.

La table des processus nous permet d'accéder à la table des régions. Les structures de régions de la table des régions contiennent des informations sur le type, les droits d'accès et la localisation (adresses en mémoire ou adresses sur disque) de la région. Seule la zone u du processus courant est manipulable par le noyau, les autres sont inaccessibles.

Tous les processus sont associés à une entrée dans la table des processus qui est interne au noyau. De plus, le noyau alloue pour chaque processus une structure appelée zone u , qui contient des données privées du processus, uniquement manipulables par le noyau. La table des processus nous permet d'accéder à la table des régions. Les structures de régions de la table des régions contiennent des informations sur le type, les droits d'accès et la localisation (adresses en mémoire ou adresses sur disque) de la région. Seule la zone u du processus courant est manipulable par le noyau, les autres sont inaccessibles.

Tous les processus sont associés à une entrée dans la table des processus qui est interne au noyau. De plus, le noyau alloue pour chaque processus une structure appelée zone u , qui contient des données privées du processus, uniquement manipulables par le noyau. La table des processus nous permet d'accéder à la table des régions. Les structures de régions de la table des régions contiennent des informations sur le type, les droits d'accès et la localisation (adresses en mémoire ou adresses sur disque) de la région. Seule la zone u du processus courant est manipulable par le noyau, les autres sont inaccessibles.

Tous les processus sont associés à une entrée dans la table des processus qui est interne au noyau. De plus, le noyau alloue pour chaque processus une structure appelée zone u , qui contient des données privées du processus, uniquement manipulables par le noyau. La table des processus nous permet d'accéder à la table des régions. Les structures de régions de la table des régions contiennent des informations sur le type, les droits d'accès et la localisation (adresses en mémoire ou adresses sur disque) de la région. Seule la zone u du processus courant est manipulable par le noyau, les autres sont inaccessibles.

Quand un processus réalise un `fork`, le contenu de l'entrée de la table des régions est dupliquée, chaque région est ensuite, en fonction de son type, partagée ou copiée (pour les fils du processus). Quand un processus réalise un `exec`, il y a libération des régions et réallocation de nouvelles régions en fonction des valeurs définies dans le nouvel exécutable.

Quand un processus réalise un `fork`, le contenu de l'entrée de la table des régions est dupliquée, chaque région est ensuite, en fonction de son type, partagée ou copiée (pour les fils du processus). Quand un processus réalise un `exec`, il y a libération des régions et réallocation de nouvelles régions en fonction des valeurs définies dans le nouvel exécutable.

Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Le contexte d'un processus est l'ensemble de

1. son état ;
2. son mot d'état : en particulier
   - La valeur des registres actifs ;
   - Le compteur ordinal.
3. les valeurs des variables globales statiques ou dynamiques ;
4. son entrée dans la table des processus ;
5. sa zone u ;
6. sa pile ;
7. les zones de code et de données.

Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Le contexte d'un processus est l'ensemble de

1. son état ;
2. son mot d'état : en particulier
   - La valeur des registres actifs ;
   - Le compteur ordinal.
3. les valeurs des variables globales statiques ou dynamiques ;
4. son entrée dans la table des processus ;
5. sa zone u ;
6. sa pile ;
7. les zones de code et de données.

Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Le contexte d'un processus est l'ensemble de

1. son état ;
2. son mot d'état : en particulier
   - La valeur des registres actifs ;
   - Le compteur ordinal.
3. les valeurs des variables globales statiques ou dynamiques ;
4. son entrée dans la table des processus ;
5. sa zone u ;
6. sa pile ;
7. les zones de code et de données.

Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Le contexte d'un processus est l'ensemble de

1. son état ;
2. son mot d'état : en particulier
   - La valeur des registres actifs ;
   - Le compteur ordinal.
3. les valeurs des variables globales statiques ou dynamiques ;
4. son entrée dans la table des processus ;
5. sa zone u ;
6. sa pile ;
7. les zones de code et de données.

Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu. Le contexte d'un processus est l'ensemble de

1. son état ;
2. son mot d'état : en particulier
   - La valeur des registres actifs ;
   - Le compteur ordinal.
3. les valeurs des variables globales statiques ou dynamiques ;
4. son entrée dans la table des processus ;
5. sa zone u ;
6. sa pile ;
7. les zones de code et de données.

Le noyau et ses variables ne font partie du contexte d'aucun processus ! L'exécution d'un processus se fait dans son contexte donné. Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte. Le noyau s'exécute alors dans le nouveau contexte.

Le noyau et ses variables ne font partie du contexte d'aucun processus ! L'exécution d'un processus se fait dans son contexte donné. Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte. Le noyau s'exécute alors dans le nouveau contexte.

Le noyau et ses variables ne font partie du contexte d'aucun processus ! L'exécution d'un processus se fait dans son contexte donné. Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte. Le noyau s'exécute alors dans le nouveau contexte.

Le noyau et ses variables ne font partie du contexte d'aucun processus ! L'exécution d'un processus se fait dans son contexte donné. Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte. Le noyau s'exécute alors dans le nouveau contexte.

Le noyau et ses variables ne font partie du contexte d'aucun processus ! L'exécution d'un processus se fait dans son contexte donné. Quand il y a changement de processus courant, il y a réalisation d'une commutation de mot d'état et d'un changement de contexte. Le noyau s'exécute alors dans le nouveau contexte.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Une interruption est un signal électrique reçu sur l'une des bornes du processeur. Ce signal est la conséquence d'un événement " extérieur " ou intérne. Une fois que le signal est détecté, il faut déterminer la cause de l'interruption. Pour cela on utilise un indicateur, pour les différentes causes : on parle alors de vecteur d'interruptions. Il existe trois grands types d'interruptions :

- externes (indépendantes du processus) telles que les interventions de l'opérateur, pannes, etc ;
- déroutements : erreur interne du processeur, débordement, division par zéro, etc (causes qui entraînent la réalisation d'une sauvegarde sur disque de l'image mémoire " core dump ") ;
- appels système : une demande d'entrée-sortie par exemple.

Suivant les machines et les systèmes un nombre variable de niveaux d'interruption est utilisé. Il est possible que plusieurs interruptions soient positionnés quand le système les consulte. C'est le niveau des différentes interruptions qui va permettre au système de sélectionner l'interruption à traiter en priorité. L'horloge est l'interruption la plus prioritaire sur un système Unix.

Suivant les machines et les systèmes un nombre variable de niveaux d'interruption est utilisé. Il est possible que plusieurs interruptions soient positionnés quand le système les consulte. C'est le niveau des différentes interruptions qui va permettre au système de sélectionner l'interruption à traiter en priorité. L'horloge est l'interruption la plus prioritaire sur un système Unix.

Suivant les machines et les systèmes un nombre variable de niveaux d'interruption est utilisé. Il est possible que plusieurs interruptions soient positionnés quand le système les consulte. C'est le niveau des différentes interruptions qui va permettre au système de sélectionner l'interruption à traiter en priorité. L'horloge est l'interruption la plus prioritaire sur un système Unix.

Suivant les machines et les systèmes un nombre variable de niveaux d'interruption est utilisé. Il est possible que plusieurs interruptions soient positionnés quand le système les consulte. C'est le niveau des différentes interruptions qui va permettre au système de sélectionner l'interruption à traiter en priorité. L'horloge est l'interruption la plus prioritaire sur un système Unix.