Systèmes d'Exploitation
Gestion des processus

Didier Verna
didier@lrde.epita.fr
http://www.lrde.epita.fr/~didier

Version 2@1.6 – 6 décembre 2004

Table des matières

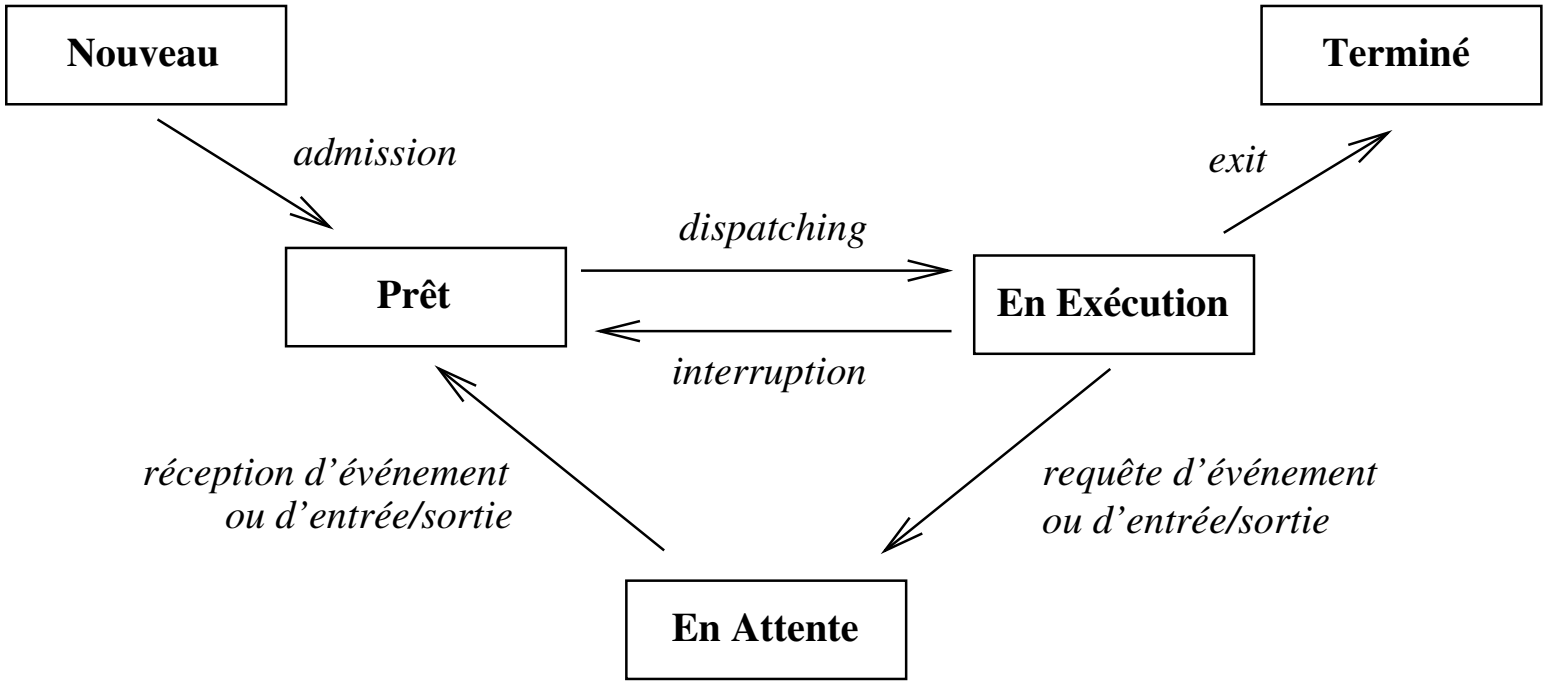
Généralités

- Un **programme** est une suite d'instructions
  (objet statique)
- Un **processus** est un programme en exécution
  (objet dynamique : programme + contexte)
- **Contexte :** Espace d'adresses (exécutable, zone de données, pile), registres (PC, SP), d'autres informations.

- Exécution simultanée de copies d'un même programme.
- Exécution simultanée de la *même* copie d'un *même* programme (« réentrance »).
- Cas intermédiaires : partager seulement le code, les données en lecture seule.

Multitâche $\Longrightarrow$ plusieurs processus exécutés simultanément $\Longrightarrow$ notion d'« état »

États d'un processus



Un processeur ne peut exécuter qu'un seul processus à la fois.

Bloc de contrôle d'un processus (PCB)

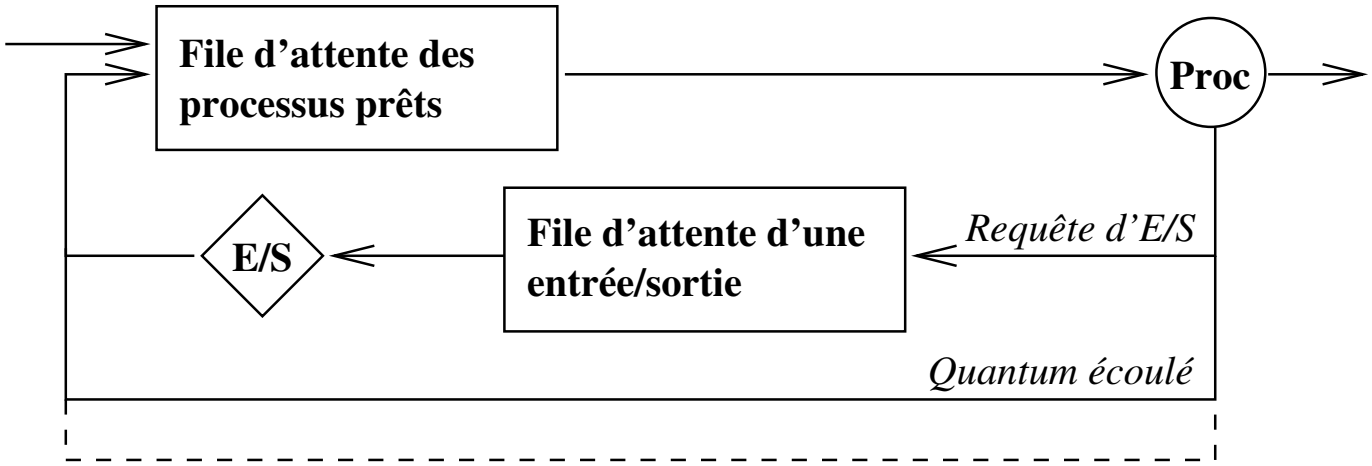
Structure de sauvegarde du contexte d'exécution d'un processus (UNIX : « U-Structure »)

- **Gestion des processus :**
  - **État :** prêt, en exécution. . .
  - **Registres :** PC, PSW. . .
  - **Identité :** PID, parent, groupe. . .
  - **Ordonnancement :** priorité, comptabilité. . .
- **Gestion de la mémoire :**
  - **Zones d'accès :** pointeurs de pile, tas, texte. . .
- **Gestion des fichiers :**
  - **E/S** : fichiers ouverts. . .
  - **Répertoires** : répertoire courant, racine. . .
  - **Identité :** utilisateur, groupe. . .

Ordonnancement des processus

Le système maintient :

- une **table de processus**,
- une **file d'attente** des processus prêts,
- des files d'attente de processus bloqués (ex. par périphérique).



Types d'ordonnanceurs

- **Long terme** :
  Présélection des tâches spoolées sur disque et mise en mémoire (traitement par lots).
  S'exécute très peu souvent.
- **Court terme** :
  Sélection des processus à exécuter parmi les processus prêts (en temps partagé).
  S'exécute très souvent.
- **Moyen terme** :
  Gestion de la mémoire auxiliaire (en temps partagé). « Swapping ».

Commutation de contexte

Échange des données concernant les processus (PCB) au moment du dispatching.

- Surcharge de travail pour le processeur.
- Vitesse liée à la complexité du système, la quantité de registres, l'existence d'instructions matérielles. . .
- Goulot d'étranglement pour les systèmes modernes (Cf. les threads).

Opérations sur les processus

Création de processus :

- **Causes :** Initialisation système, création par un autre processus, requête utilisateur...(UNIX : `fork`, Win32 : `CreateProcess`).
- **Nature :** processus interactifs, tâches de fond (background), démons...(UNIX : `ps`, Windows : `Ctrl-Alt-Del`).
- **Types :** clônage du processus parent, nouvelle tâche (UNIX : `exec`).

Processus coopératifs :

- **Efficacité :** répartition du travail en sous-tâches concurrentes
- **Modularité :** répartition du travail en sous-tâches logiques

Terminaison de processus

Causes :

- Terminaison (a)normale mais volontaire.
  UNIX : `exit`, Windows : `ExitProcess`.
- Terminaison anormale involontaire (bugs : Division par 0, accès mémoire illégal. . .).
  Possibilité de paraméter le comportement (signaux UNIX).
- Terminaison normale involontaire.
  UNIX : `kill`, Windows : `TerminateProcess`.

Processus coopératifs :

- **Synchronisation :** bloquage et attente de la terminaison du fils (UNIX : `wait`).
- **Communication :** renvoi d'une valeur de terminaison.

Hiérarchies de processus

Association permanente entre parent et enfant

- **Windows :** pas de hiérarchie. Le « process handle » peut circuler.
- **UNIX :**
  - Arborescence sous `init`.
  - Notion de « groupe de processus ». Réception des signaux par groupe.
  - Exception : `nohup`.
- **Terminaison en cascade :** un processus ne peut pas survivre à son parent.

Le multithreading

Les threads répondent aux besoins suivants :

- fournir un parallélisme intra-processus,
- partager les données d'un processus entre plusieurs fils d'exécution,
- amoindrir le coût de la commutation de contexte.

- **Thread / LWP :** (Light Weight Processus) état, ensemble de registres (dont PC) et pile. Partage des autres ressources entre tous les threads d'un même processus.
- **Tâche / HWP :** (High Weight Processus) ensemble de threads.
  Processus traditionnel = tâche à un seul thread.

Fonctionnement des threads :

- Comportement similaire à celui des processus : création, terminaison, notion d'état, synchronisation. . .
- Problèmes nouveaux : héritage des threads après `fork` (threads bloqués), concurrence d'accès aux ressources partagées. . .

Threads utilisateur

- Implémentés par une bibliothèque en espace utilisateur.
- Une table de threads par processus.

Avantages :

- Utilisable au dessus d'un système non multithreadé.
- Commutation de contexte très rapide (pas de « kernel trapping »).
- Algorithmes d'ordonnancement personalisable.

Inconvénients :

- Appels système bloquants : versions non bloquantes, wrappers utilisant `select`...
- Threads monopolisant le CPU : `yield`, requête d'alarme...
- L'intérêt des threads réside justement là où il y a souvent bloquage.
  $\Longrightarrow$ Le noyau est là pour ça !

Threads noyau

- Une table de threads en plus de la table de processus.
- Tout appel bloquant est implémenté par appel système.

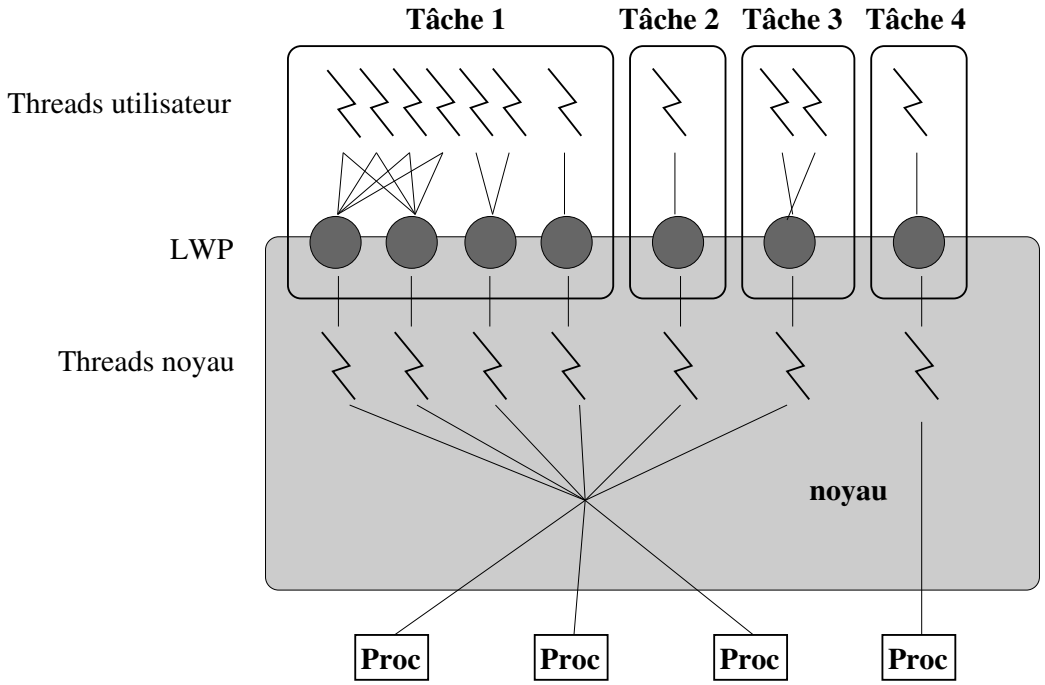
Avantages :

- Facilité de conception des applications.
- Pas de nécessité de procédures supplémentaires non bloquantes.

Inconvénients :

- Coût de gestion des threads (recyclage).
- Coût des appels bloquant (interruptions).

Approches hybrides (ex. Solaris 2)



Scheduler activation :

- « upcall » vers l'ordonnanceur logiciel (userland) quand un thread se (dé)bloque.
- Rupture avec le modèle en couches.

Transition vers le multithreading

Des problèmes nouveaux :

- Variables globales (programmes ou système) :
  - les interdire,
  - fournir des valeurs locales par thread (ex. `errno`).
- Signaux : qui les récupère ?
- Réentrance des fonctions (bibliothèques systèmes) :
  - mettre des drapeaux d'exclusion mutuelle (attention au parallélisme),
  - réécrire les bibliothèques.

Communication inter-processus

Les processus coopératifs ont besoin de communiquer
(mémoire partagée ou envoi de messages)

- L'envoi de message permet de se passer du partage de mémoire.
- De nombreux systèmes proposent cependant les deux fonctionnalités.
- IPC : Inter-Process Communication.

Problématique :

- Comment établir une liaison ?
- Combien de processus par liaison ?
- Combien de liaison par processus ?
- etc.

Communication directe

Nommage explicite du destinataire ou de l'émetteur

- Liaison automatique créée par le système
- Une seule paire de processus par liaison
- Liaison uni ou bidirectionnelle

- **Liaison symétrique** : `send (P, msg); recv (P, msg);`
- **Liaison asymétrique** : `send (P, msg); recv (id, msg);`

$\Longrightarrow$ Nécessité de connaître les noms des processus.
Problème en cas de changement de nom.

Communication indirecte

Envoi et réception sur des *ports*

```
send (A, msg); recv (A, msg);
```

- Liaison établie à condition que les processus partagent un même port
- Plus de deux processus par liaison
- Liaison uni ou bidirectionnelle

$\Longrightarrow$ Problème à gérer : multiples réceptions simultanées sur un même port (se restreindre à deux processus, limiter les types d'accès en lecture ou écriture...).

Bufferisation

Caractérisation de la capacité d'une liaison

- **Capacité 0** : aucun message en attente. Nécessité d'une synchronisation pour échanger des messages (« rendez-vous » / « hand shaking »).
- **Capacité limitée** : l'émetteur peut être retardé si le buffer est plein.
- **Capacité illimitée** : l'émetteur n'est jamais retardé.

Pour des capacités non nulles, la communication est **asynchrone** :

- Un émetteur ne sait pas si son message est reçu.
- Synchronisation possible par « acquittement » (acknowledgement).
- Synchronisation possible en rendant `send` bloquant jusqu'à l'arrivée d'un accusé de réception.

Cas pathologiques

- **Terminaison** : P attend (ou envoie) un message de (ou vers) un processus terminé. Le système doit le terminer ou lui notifier le problème.
- **Messages perdus** :
  - Vérification sous la responsabilité de l'émetteur lui-même.
  - Renvoi (ou notification à l'émetteur) sous la responsabilité du système.
  - Détection par temporisation.
  - Pas de détection.
- **Messages bruités** : (bitflip) idem. Détection par contrôle de parité.