Master M1 Informatique, UJF Grenoble
TD-TP Processus sous Unix/Linux

Vincent Danjean, Guillaume Huard, Arnaud Legrand
Vania Marangozova-Martin, Jean-François Méhaut

année 2011-2012

Résumé

Cette fiche ne sera traitée que cette semaine. Vous traiterez en dehors de la séance les exercices et questions non résolues. Les objectifs de cette première séance de TD sont :

- Manipulation des processus et tubes sous Linux
- Primitives `fork` et `pipe`
- Utilisation de segments de mémoire partagée
- Synchronisation par attente active

Les exercices portant la mention "Travail personnel" figurant en fin de feuille de TD sont optionnels : le temps imparti à une séance de TD ne permettra sans doute pas de les traiter durant celle-ci et les évaluations de l'UE ne porteront pas sur les notions qu'ils permettent d'aborder. Il sont donnés à titre de travail personnel. Si vous ne parvenez pas à le traiter seul n'hésitez pas à demander un peu d'aide à vos enseignants.

1 Un petite exercice pour se mettre en condition

On souhaite écrire une petite application constituée de n processus où chacun des processus va générer une nombre aléatoire et afficher ce nombre aléatoire à l'écran. Voici un exemple d'exécution de cette applicaton avec 6 processus.

```
$ ring1 6
processus pid 25387 node 2 val = 1430826605
processus pid 25388 node 3 val = 48523501
processus pid 25389 node 4 val = 822619539
processus pid 25390 node 5 val = 1591287596
processus pid 25385 node 0 val = 2047288621
processus pid 25386 node 1 val = 1731323093
$
```

L'application est donc constituée de 6 processus. Le nombre de processus de l'application est passé en paramètre du programme. Les processus sont identifiés par leur pid (retourné par le `fork`), un numéro de nœud (numéro d'ordre dans la création) et un nombre aléatoire généré par la fonction `rand()`. Pour avoir plus d'informations sur cette fonction, vous pouvez les obtenir avec le manuel (`man rand`). Prenez garde à ce que les séquences de nombres aléatoires générées par chaque processus soient différentes (voir la fonction `srand()`).

2 Un peu plus difficile

Il s'agit maintenant de reprendre l'exemple précédent et de déterminer le processus qui a généré aléatoirement le plus grand entier. En terme algorithmique, il s'agit d'un mécanisme d'élection. Pour implémenter ce mécanisme d'élection, nous allons utiliser les tubes de communication et connecter les

processus avec une topologie de type anneau. Le processus 0 va être connecté avec le processus 1 par un tube, le processus 1 avec le processus 2 par un autre tube, le processus 2 avec le processus 3 et ainsi de suite... Le dernier processus de l'anneau doit reboucler sur le processus 0.

Lorsque les processus et les tubes sont créés, le processus 0 envoie sa valeur sur l'anneau (vers le processus 1) qui compare la valeur reçue du processus 0 par rapport à sa valeur et envoie au processus 1 la valeur la plus grande et chaque processus de l'anneau va appliquer le même traitement. A la fin de cette phase, le processus 0 doit recevoir sur son tube la plus grande valeur.

```
$ ring2 6
processus pid 26603 node 1 val = 982695543
processus pid 26604 node 2 val = 679092432
processus pid 26605 node 3 val = 1441867048
processus pid 26606 node 4 val = 1135529882
processus pid 26607 node 5 val = 1906462017
processus pid 26602 node 0 val = 208930720
the winner is 1906462017 pid 26607 node 5
$
```

Sur cet exemple, c'est le processus 5 qui a généré le plus grand nombre. Le processus 0 envoie son nombre au processus 1 et on fait ainsi un tour de l'anneau. A la fin, le processus 0 affiche que le gagnant est le processus 5.

3 Mais qui est le gagnant ?

Pour cet exercice, on souhaite maintenant informer l'ensemble des processus du nom du gagnant. Sur l'exercice précédent, seuls les processus 0 et 5 connaissaient l'identité du processus gagnant. Pour cet exercice, le processus 0 doit faire passer sur l'anneau l'identité du gagnant. Le gagnant saura ainsi qu'il était le gagnant !

```
$ ring3 6
processus pid 27739 node 1 val = 161183994
processus pid 27740 node 2 val = 925511728
processus pid 27741 node 3 val = 1709276223
processus pid 27742 node 4 val = 1410188147
processus pid 27743 node 5 val = 1099100972
processus pid 27738 node 0 val = 1532950038
Node 0 the winner is 1709276223 pid 27741 node 3
Node 1 the winner is 1709276223 pid 27741 node 3
Node 2 the winner is 1709276223 pid 27741 node 3
Node 3 the winner is 1709276223 pid 27741 node 3
Node 4 the winner is 1709276223 pid 27741 node 3
Node 5 the winner is 1709276223 pid 27741 node 3
$
```

Sur cet exemple, on voit que c'est le processus 3 qui est gagnant. Le processus 0 est le premier à le savoir, c'est ensuite au tour du processus 1 et ainsi de suite...

4 Vous n'aimez pas les tubes, vous allez peut-être aimer les segments de mémoire partagée !

Vous avez peut-être eu quelques difficultés à mettre en place l'anneau qui allait interconnecter vos processus. Nous vous proposons maintenant de mettre en place cette communication entre les processus par un segment de mémoire partagée. Un segment de mémoire partagée est une zone mémoire qui va pouvoir être partagée entre plusieurs processus. Dès qu'un processus écrit dans cette zone mémoire, la modification va être automatiquement visible par les autres processus de l'application.

Pour créer un segment, il faut d'abord demander au système d'allouer un espace mémoire partagé (opération `shmget`). Le système nous fournit alors l'identifiant de cet espace mais celui-ci n'est pas

accessible depuis l'espace mémoire du processus. Il faut donc attacher le segment à l'espace mémoire du processus (opération `shmat`). Ce mécanisme en deux temps permet, au besoin, de créer et partager un segment après le démarrage des processus : il suffit qu'un processus crée le segment et diffuse l'identifiant correspondant pour que tous les processus puissent l'attacher à leur espace mémoire. Pour ce TD, nous pourrons nous limiter à une utilisation plus simple : le premier processus crée et attache le segment à son espace mémoire avant de créer les autres processus. Le segment est alors visible par tout le monde.

Une fois l'utilisation du segment terminée, il ne faut pas oublier de le détacher (opération `shmdt`) et surtout de le libérer afin que le système puisse réccupérer la mémoire associée (opération `shmctl` avec la commande `IPC_RMID` passée en paramètre).

Voici un extrait de code illustrant la création et la destruction d'un segment de mémoire partagée :

```
void handle_error(char *function_name, int return_value) {
    char message[128] = "Error in function ";
    if (return_value == -1) {
        perror(strcat(message, function_name));
        exit(1);
    }
}

void *create_shared_memory(size_t size, int *shmid) {
    void *ptr;

    *shmid = shmget(IPC_PRIVATE, size, IPC_CREAT | 0666);
    handle_error("shmget", *shmid);
    ptr = shmat(*shmid, NULL, 0);
    handle_error("shmat", (int) ptr);
    return ptr;
}

void delete_shared_memory(void *ptr, int shmid) {
    int cr;
    cr = shmdt(ptr);
    handle_error("shmdt", cr);
    cr = shmctl(shmid, IPC_RMID, NULL);
    handle_error("shmctl", cr);
}
```

Le segment va donc être partagé par l'ensemble des processus. Chaque processus va écrire ses informations dans sa zone. Il va ensuite lire ce que les autres processus auront écrits. Si les autres processus n'ont pas encore écrit leurs valeurs, le processus va devoir attendre que les autres processus aient écrit avant de pouvoir décider qui a remporté cette élection. Chaque processus prend la décision à partir des données écrites par les autres processus.

5 Travail personnel : lister les processus du système

L'objectif de cet exercice est de lister les processus du système un peu de la même manière que le font les commandes `ps`, `top` ou bien `htop`. Pour réaliser cet exercice, nous nous appuierons sur le contenu du répertoire `proc`.

```
$ ls /proc
1 1630 2024 2415 2538 2717 2926 2977 3032 332 5015 860 diskstats kcore self
11980 17666 2027 2449 2553 2771 2928 298 3040 3337 5017 861 dma kmsg slabinfo
1291 17667 203 2459 2569 2772 2943 299 3048 334 6 9 driver loadavg stat
1293 19144 204 2499 2578 2773 2947 3 305 337 690 966 execdomains locks swaps
1295 19190 205 2512 2590 2774 2948 3006 3094 3449 7 acpi fb meminfo sys
1297 19330 206 2513 2604 2775 3096 3454 749 asound filesystems misc sysvipc
1299 19352 207 2520 2631 2776 2964 301 3106 3783 762 buddyinfo fs modules tty
13 19360 2120 2527 2640 2867 2967 3012 3107 3803 775 bus interrupts mounts uptime
```

14 19559 2131 2528 2670 2911 2968 3017 3140 4 8 cmdline iomem mtrr version
15 1986 2132 2533 2705 2914 2969 3022 3239 457 857 cpuinfo ioports [**net** vmstat
158 1996 2155 2535 2707 2915 297 4992 858 crypto **irq** partitions zoneinfo
1602 2 2163 2537 2713 2917 2974 3027 3313 5 859 devices kallsyms **scsi**
$

Les noms en gras sont des noms de répertoires. Les noms de répertoires qui sont des nombres correspondent aux PID des processus. Pour chaque processus, il y a donc un répertoire qui a comme nom la chaÓne de caractères correspondant au PID du processus.

Pour avoir la liste des processus du système, il suffit donc de lister le contenu de ce répertoire et prendre les noms de répertoires correpondant à des nombres. Il faut pour cela utiliser les fonctions `opendir`, `readdir`, `closedir`.

Nous allons maintenant consulter le contenu d'un répertoire correspondant à un processus.

```
$ ls 19352
dr-xr-xr-x   4 mehaut mehaut 0 2007-10-04 08:59 .
dr-xr-xr-x 149 root   root   0 2007-10-03 19:52 ..
-r--------   1 mehaut mehaut 0 2007-10-04 09:34 auxv
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 cmdline
lrwxrwxrwx   1 mehaut mehaut 0 2007-10-04 09:34 cwd -> /home/mehaut/CSE/anneau
-r--------   1 mehaut mehaut 0 2007-10-04 09:34 environ
lrwxrwxrwx   1 mehaut mehaut 0 2007-10-04 08:59 exe -> /usr/bin/xpdf.bin
dr-x------   2 mehaut mehaut 0 2007-10-04 09:34 fd
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 maps
-rw-------   1 mehaut mehaut 0 2007-10-04 09:34 mem
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 mounts
-r--------   1 mehaut mehaut 0 2007-10-04 09:34 mountstats
-rw-r--r--   1 mehaut mehaut 0 2007-10-04 09:34 oom_adj
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 oom_score
lrwxrwxrwx   1 mehaut mehaut 0 2007-10-04 09:34 root -> /
-rw-------   1 mehaut mehaut 0 2007-10-04 09:34 seccomp
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 smaps
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 stat
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 statm
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 status
dr-xr-xr-x   3 mehaut mehaut 0 2007-10-04 09:34 task
-r--r--r--   1 mehaut mehaut 0 2007-10-04 09:34 wchan

$
```

On retrouve donc dans ce répertoire les principales informations que conserve le noyau du système pour chaque processus. On peut donc y voir par exemple :

- le lien `exe` pour le nom du programme qu'exécute le processus,
- le nom du répertoire `cwd` courant où se trouve le processus,
- le fichier `environn` avec les variables d'environnement,
- le fichier `maps` qui donne une description de l'espace de mémoire virtuelle,
- le répertoire `fd` où on retrouve l'ensemble des descripteurs de fichiers qui ont été ouverts,
- et bien d'autres choses qu'on vous laisse découvrir...

A partir de ces informations, vous pouvez essayer de lister l'ensemble des processus du système avec pour chaque processus le nom du programme qu'il exécute. Vous pourrez ensuite compléter avec les informations trouvées dans la hiérarchie pour vous rapprocher de ce que donne la commande `ps`.