

Création de processus

Fork, execl, system, atexit, pipe

UV SR01

Dr. Hicham Lakhlef

Laboratoire Heudiasyc (UMR UTC-CNRS 7253)

Université de Technologie de Compiègne

France

hlakhlef AT utc.fr

A2022

Création de processus

1 Création de processus

- Introduction
- Créat. fork()
- Recouv. exec()
- CMD : system
- Exercice

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

2 Terminaison

- Caractéristiques
- Processus zombies
- Code de retour

3 Pipes

- Caractéristiques
- Pipes anonymes
- Pipes nommés

Sommaire

Création de processus

1 Création de processus

- Introduction
- Créat. fork()
- Recouv. exec()
- CMD : system
- Exercice

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

2 Terminaison

- Caractéristiques
- Processus zombies
- Code de retour

3 Pipes

- Caractéristiques
- Pipes anonymes
- Pipes nommés

Introduction

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

■ Processus dans le système

- Un processus (process) est un programme en cours d'exécution dans un ordinateur \Rightarrow **entité dynamique**
- *init* est l'ancêtre de tous les processus, c'est le premier lancé et c'est pour ça qu'il est identifié par le numéro 1
- Les autres processus sont créés par des processus
- La commande *ps tree* affiche les processus en cours d'exécution sous format hiérarchique (arbre)

■ Génétique des processus

- Opération de création **fork()** (Appel système)
- Opération de recouvrement **exec()**
- Héritage de données
- Facilite la mise en place d'une communication entre père et fils

Création de processus avec fork()

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- l'appel système fork()
 - Prototype : `pid_t fork(void)`
 - Crée un nouveau processus identique par duplication du processus appelant
 - Appel système : accéder aux ressources matérielles
 - Retour du `PID=fork() = 0` pour le processus fils
 - Retour du `PID=fork() > 0` pour le processus père, qui reçoit le PID du processus fils créé

ex_fork.c : exemple de fork (debut)

```
#include <sys/types.h> /* Types pid_t... */
#include <unistd.h> /* fork()... */
#include <stdio.h> /* printf... */
#include <stdlib.h> /* EXIT_FAILURE... */
#define DURATION 15
int main()
{
    pid_t pid_fils; int i;
```

Création de processus avec fork()

(suite exemple)

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

```
switch ( pid_fils = fork() ) {
case (pid_t)-1:
/* Retour du fork() = -1 => echec (eg. manque de ressources) */
perror("main/fork");
return EXIT_FAILURE;
case (pid_t)0: /* Retour du fork() = 0 => processus fils */
for(i=DURATION; i>0; i--) {
printf("Fils : je suis vivant pour encore %d secondes.\n", i);
fflush(stdout); sleep(1);
}
return EXIT_SUCCESS;
default: /* Retour du fork() != 0 donc c'est le pere */
for(i=DURATION; i>0; i--) {
printf("Pere : je suis vivant pour encore %d secondes.\n", i);
fflush(stdout); sleep(1);
}
}
return EXIT_SUCCESS;
}
```

Création de processus avec fork()

(suite exemple)

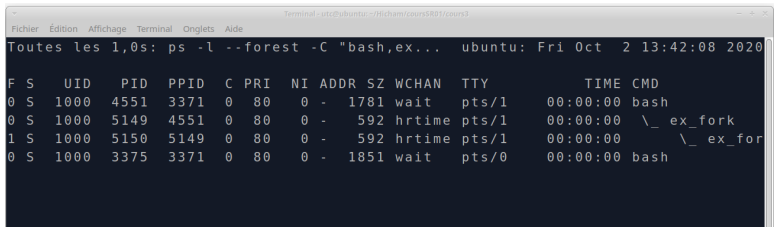
- Dans un premier terminal tapez la commande suivante :

watch -n 1 'ps -l --forest -C "bash,ex_fork"'

- watch : exécute la commande périodiquement
- ps -l : format long
- ps forest : affiche l'arborescence des processus
- ps -C "bash,fork" : uniquement les commandes bash et fork

- Dans un second terminal tapez la commande suivante :

./ex_fork



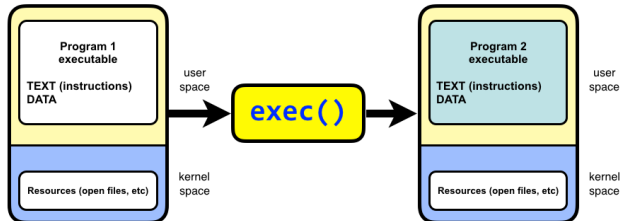
```
Terminal - ubuntu: ~/Hicham/coursSR07/cours3
Fichier  Édition  Affichage  Terminal  Onglets  Aide
Toutes les 1,0s: ps -l --forest -C "bash,ex...  ubuntu: Fri Oct  2 13:42:08 2020

 F S      UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S    1000    4551   3371  0  80   0  -  1781 wait  pts/1      00:00:00 bash
0 S    1000    5149   4551  0  80   0  -   592 hrtime pts/1      00:00:00 \_ ex_fork
1 S    1000    5150   5149  0  80   0  -   592 hrtime pts/1      00:00:00 \_ ex_for
0 S    1000    3375   3371  0  80   0  -  1851 wait  pts/0      00:00:00 bash
```

Recouvrement d'un processus

■ Famille de fonctions **exec**

- Un processus fils créé peut remplacer son code de programme par un autre programme
- Tous les appels système **exec** remplacent le processus courant par un nouveau processus construite à partir d'un fichier ordinaire exécutable
- Les segments de texte et de données du processus sont remplacés par ceux du fichier exécutable
- Ne retourne une valeur (-1) que si l'appel système échoue
- En cas de succès, le code qui suit n'est pas exécuté (il a été recouvert)



Recouvrement d'un processus

Les fonctions exec

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

```
#include <unistd.h>
```

- `int execl(const char *path, const char *argv, ...);`
- `int execv(const char *path, const char *argv[]);`
- `int execlp(const char *path, const char *argv, const char *envp[]);`
- `int execlp(const char *file, const char *argv, ...);`
- `int execvp(const char *file, const char *argv[]);`
- Exemple avec `execv` : `execv("/bin/ls", "-l", NULL);`

Recouvrement d'un processus

Exemple

Création de
processus

Création de
processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

execv(): exemple

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

main()
{
    pid_t pid;
    char *const arguments[]={"/bin/ls", "-l", NULL};

    if ((pid = fork()) == -1)
        perror("erreur de fork()");
    else if (pid == 0)
    {
        execv("/bin/ls", arguments);
        printf("Retour non attendu. Doit tre une erreur execv\n");
    }
}
```

Appel d'une commande avec system()

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Prototype : *int system(const char *commande);*
- Cette fonction permet de lancer l'exécution d'une commande sur le système d'exploitation hôte : un nouveau processus est lancé pour chaque appel à cette fonction
- L'appel à cette fonction bloquera le thread en cours jusqu'à que la commande lancée finisse
- Si la sortie standard de votre programme est redirigée sur une console ou sur un fichier, vous pourrez y aussi retrouver les resultats produits par la commande exécutée par l'appel système
- Exécute une commande sans recouvrement.

Appel d'une commande avec system()

Exemple

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

ex_system.c

```
#include <stdio.h> /* printf... */
#include <stdlib.h> /* EXIT_FAILURE, system... */
#include <unistd.h> /* execl... */

int main(int argc, char **argv) {
    int i, res;
    if(argc < 2){
        printf("\n Usage : %s commande (eg. /bin/cat)\n", argv[0]);
        return EXIT_SUCCESS;
    }
    printf("\n Processus %d", getpid());
    printf("\n--- Execution de la commande %s ---\n", argv[1]);
    if ( system(argv[1]) == -1 ){
        perror("main/system");
        return(EXIT_FAILURE);
    }
    printf("\n--- Fin ---\n");
    return(EXIT_SUCCESS);
}
```

Création de Processus

Exercice : implémenter system()

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Implémenter system()
 - Récupérer la commande à lancer via les arguments de la ligne de commande
 - Combiner fork() et exec()
 - Création d'un processus fils avec fork()
 - Recouvrement du fils avec exec() qui lance la commande

Création de Processus

Exercice : implémenter system()

Création de processus

solution (1/2)

```
#include <sys/types.h> /* Types pid_t... */
#include <unistd.h> /* fork()... */
#include <stdio.h> /* printf... */
#include <stdlib.h> /* EXIT_FAILURE... */
#define DURATION 15

int main(int argc, char **argv){
    pid_t pid_fils;
    int i;
    if(argc < 2)
    {
        printf("\n Usage : %s commande (eg. /bin/cat)\n", argv[0]);
        return EXIT_SUCCESS;
    }

    switch ( pid_fils = fork() ) {
        case (pid_t)-1:
            /* Retour du fork() = -1 => echec (eg. manque de ressources) */
            perror("main/fork");
            return EXIT_FAILURE;
```

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Création de Processus

Exercice : implémenter system()

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

solution (2/2)

```
case (pid_t)0:
    /* Retour du fork() = 0 => processus fils */
    printf("\n Processus fils %d de pere %d", getpid(), getppid());
    printf("\n --- Execution de la commande %s ---\n", argv[1]);
    execl(argv[1], argv[1], NULL);
    perror("main/execl");
    return EXIT_FAILURE;
default:
    /* Retour du fork() != 0 donc c'est le pere */
    for(i=DURATION; i>0; i--)
    {
        printf("\n Proc. pere vivant pour encore %d secondes.", i);
        fflush(stdout);
        sleep(1);
    }
    return EXIT_SUCCESS;
}
```

Création de Processus

Exercice : implémenter system()

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Dans un premier terminal :
*watch -n 1 'ps -l - -forest -C
"exo_system,bash,cat,ls,upstart"'*
- Dans un second terminal :
*./exo_system "/bin/ls"
./exo_system "/bin/cat"*
- Permet d'observer la fin du processus fils, avant/après la fin du processus père

Création de processus

Création de processus

Introduction
Créat. fork()
Recouv. exec()
CMD : system
Exercice

Terminaison

Caractéristiques
Processus zombies
Code de retour

Pipes

Caractéristiques
Pipes anonymes
Pipes nommés

1 Création de processus

- Introduction
- Créat. fork()
- Recouv. exec()
- CMD : system
- Exercice

2 Terminaison

- Caractéristiques
- Processus zombies
- Code de retour

3 Pipes

- Caractéristiques
- Pipes anonymes
- Pipes nommés

Terminaison d'un processus

Introduction

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Un processus interagit avec le système et/ou d'autres processus
 - Ouverture, modification de fichiers
 - Envoi de messages
 - ...
- Un processus peut se terminer inopinément
 - Terminaison initiée par l'utilisateur du programme
 - Terminaison par une commande shell eg. kill
 - Terminaison par le système
- Terminaison propre
 - Gérer le changement d'état des ressources
 - Garder la cohérence des fichiers
 - ...
- Un père doit veiller sur ses fils
 - Ne pas laisser de processus fils à l'état zombi
 - Sinon encombrement des tables du système

Caractéristiques

Installation d'une fonction avec atexit() (1/2)

- **atexit()** permet d'installer une fonction qui sera exécutée sur terminaison normale du programme
 - Prototype : `int atexit(void (*function)(void));`
 - `void (*function)(void)` : pointeur de fonction de type `void function(void)`
 - Programme l'appel d'une fonction lorsque le processus se termine normalement.
 - Empilement des fonctions si plusieurs appels

atexit(): exemple (debut)

```
void terminer(void)
{
    printf("\nTerminaison du processus %d (fonction \"terminer\")", P
}

void quitter(void)
{
    printf("\nFin (fonction \"quitter\")\n");
}
```

Caractéristiques

Installation d'une fonction avec atexit() (2/2)

Création de processus

atexit(): exemple (fin)

```
int main(int argc, char **argv){
    int i;
    if(argc < 2){
        printf("\n Usage : %s delais\n", argv[0]);
        return EXIT_SUCCESS;
    }
    pid = getpid();
    if( atexit(quit) != 0 || atexit(terminer) != 0){
        perror("main/atexit");
        return EXIT_FAILURE;
    }
    for(i=atoi(argv[1]); i>0; i--){
        printf("\nLe proc. %d vivant pour encore %d s.", pid, i);
        fflush(stdout);
        sleep(1);
    }
    return EXIT_SUCCESS;
}
```

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Caractéristiques

Terminaisons imprévues (1/3)

- Les terminaisons imprévues correspondent à la réception des signaux comme SIGQUIT, SIGTERM, SIGINT...
- Utilisation de handlers (gestionnaire) pour modifier le traitement des signaux

Exemple de terminaisons imprévue (1/3)

```
#include <signal.h> /* sigaction... */
#include <stdio.h> /* printf... */
#include <stdlib.h> /* EXIT_FAILURE... */
#include <sys/types.h> /* Types pid_t... */
#include <unistd.h>

pid_t pid;

void terminer(void) {
    printf("\nTerminaison du processus %d", pid);
}

void quitter(void)
{
    printf("\nFin\n");
    exit(EXIT_SUCCESS);
}
```

Caractéristiques

Terminaisons imprévues (2/3)

Exemple de terminaisons imprévue (2/3)

```
void interceptor(int n){
    int i;
    printf("\nRception du signal %d (INT=%d, TERM=%d,
QUIT=%d)", n, SIGINT, SIGTERM, SIGQUIT);
    switch (n) {
        case SIGTERM: terminer();
        case SIGINT:
        case SIGQUIT: quitter();
    }
    printf("\nFin du handler");
}

int main(int argc, char **argv) {
    int i; struct sigaction S;
    if(argc < 2){
        printf("\n Usage : %s delais\n", argv[0]);
        return EXIT_SUCCESS;
    }
```

Caractéristiques

Terminaisons imprévues (3/3)

Exemple de terminaisons imprévue (3/3)

```
pid = getpid();
S.sa_handler = interceptor;
if( sigaction(SIGTERM, &S, NULL) != 0 || \
    sigaction(SIGQUIT, &S, NULL) != 0 || \
    sigaction(SIGINT, &S, NULL) != 0)
{
    perror("sigaction");
    exit(EXIT_FAILURE);
}
for(i=atoi(argv[1]); i>0; i--)
{
    printf("\nLe processus %d est vivant pour encore %d s.", pid, i);
    fflush(stdout);
    sleep(1);
}
return EXIT_SUCCESS;
}
```

Processus zombis (1/4)

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombis

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Lorsqu'un processus se termine (pour une raison quelconque), le système d'exploitation ne le supprime pas immédiatement du système
- Le processus reste jusqu'à ce qu'il soit "récupéré" par le parent
- Lorsque le parent récupère un processus enfant, le système d'exploitation donne au parent le statut de sortie d'enfant et nettoie l'enfant
- Un processus terminé qui n'a pas été récupéré est appelé **processus zombie**

Processus zombis (2/4)

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombis

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

■ Programme d'illustration

- Selon la valeur des arguments, le processus fils termine avant ou après le père.
- Le père appelle régulièrement la commande ps pour afficher l'état du processus fils.

Processus zombis : exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char **argv){
    int i, delais_fils, delais_pere;
    pid_t pid_fils;
    char commande[128];
    if(argc < 3) {
        printf("\n Usage : %s delais_fils delais_pere\n", argv[0]);
        return EXIT_SUCCESS; }
    delais_fils = atoi(argv[1]);
    delais_pere = atoi(argv[2]);
```

Processus zombies (3/4)

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Processus zombies : exemple (suite)

```
switch ( pid_fils = fork() ) {
    case (pid_t)-1:
        perror("main/fork");
        exit(EXIT_FAILURE);
    case (pid_t)0: /* processus fils */
        for(i=delais_fils; i>0; i=i-2) {
            printf("--> Le fils est vivant pour encore %d secondes.\n", i);
            fflush(stdout);sleep(1);
        }
        exit(EXIT_SUCCESS);
    default: /* processus pere */
        snprintf(commande, 128, "ps %d", pid_fils);
        for(i=delais_pere; i>0; i=i-2) {
            printf("Le pere est vivant pour encore %d secondes.\n", i);
            system(commande);fflush(stdout);
            sleep(1);
        }
}
```

Processus zombies (4/4)

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Dans un premier terminal :
`watch -n 1 'ps -l -forest -C "ex-zombie,bash,upstart"'`
- Dans un second terminal
 - Exemple 1 : création d'un zombi `./ex-zombie 6 60`
 - Exemple 2 : fils orphelin rattaché `./ex-zombie 60 6`

Code de retour

signal SIGCHLD

- Détection de la fin d'un fils
 - Utilisation de sigaction()
 - Intercepter le signal SIGCHLD

exemple (1/2)

```
#include <signal.h> /* sigaction... */
#include <sys/types.h> /* Types pid_t... */
#include <unistd.h> /* fork()... */
#include <stdio.h> /* printf... */
#include <stdlib.h> /* EXIT_FAILURE... */
#define CODE_RETOUR_FILS 4
pid_t pid, pid_fils;
void interceptor(int n)
{
    printf("\nProc %d : rception du signal %d (SIGCLD=%d)\n\n", pid,
}
```

Code de retour

signal SIGCHLD

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

system(): exemple (2/2)

```
int main(int argc, char **argv){
    int i, delais_fils, delais_pere;
    struct sigaction S;
    if(argc < 3)
    {
        printf("\n Usage : %s delais_fils delais_pere\n", argv[0]);
        return EXIT_SUCCESS;
    }
    delais_fils = atoi(argv[1]);
    delais_pere = atoi(argv[2]);
    S.sa_handler = interceptor;
    pid = getpid();
    if( sigaction(SIGCHLD, &S, NULL) != 0 ){
        perror("sigaction");
        exit(EXIT_FAILURE);
    }
```

Code de retour

signal SIGCHLD

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

```
switch ( pid_fils = fork() ) {
case (pid_t)-1: /* echec (eg. manque de ressources) */
perror("main/fork");
return EXIT_FAILURE;
case (pid_t)0: /* Retour du fork() = 0 => processus fils */
pid = getpid();
for(i=delais_fils; i>0; i=i-2) {
printf("\n Le fils %d:vivant pour encore %d secondes.",pid,i);
fflush(stdout); sleep(2);
}
return CODE_RETOUR_FILS;
default: /* Retour du fork() != 0 donc c'est le pere */
for(i=delais_pere; i>0; i=i-2) {
printf("\n LE pre %d:vivant pour encore %d secondes.",pid,i);
fflush(stdout); sleep(2);
}
}
printf("\n"); return EXIT_SUCCESS;
}
```

- Dans un premier terminal
`watch -n 1 'ps -l --forest -C"ex-forksig,bash,upstart"'`
- Dans un second
`./ex-forksig 6 12`
`./ex-forksig 12 6`
- Dans un troisième terminal `kill -CHLD pid_pere`

Code de retour

signal SIGCHLD

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Récupération de l'état d'un processus fils
 - Attente de la terminaison d'un fils avec waitpid()
 - Prototype waitpid(pid_t pid, int *status, int options);
 - Options
 - WNOHANG : attente non bloquante si pas de fils
 - WUNTRACED : attente terminaison ou processus stoppé
 - WCONTINUED : attente réception SIGCONT par le fils
- Décodage du statut du fils avec des macros
 - WIFEXITED(status)
 - WEXITSTATUS(status)
 - WIFSIGNALED(status)
 - WTERMSIG(status)
 - WCOREDUMP(status)
 - WIFSTOPPED(status)
 - WSTOPSIG(status)
 - WIFCONTINUED(status)

Code de retour

signal SIGCHLD

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

```
void interceptor(int n)
{
    int i, st_files;
    printf("Proc %d:reception du signal %d (SIGCLD=%d)\n",pid,n,SIGCLD);
    if (waitpid(pid_files,&st_files,WUNTRACED | WCONTINUED)==-1)
    {perror("interceptor/waitpid");}
    else
    {
        if(WIFEXITED(stat_files) != 0 ){
            printf("Fils:fin normale,code retour %d\n",WEXITSTATUS(st_files));
        }
        if(WIFSIGNALED(stat_files) != 0 ){
            printf("\nFils:fin via signal %d non intercepte\n",WTERMSIG(st_files));
        }
        if( WIFSTOPPED(stat_files) != 0 ){
            printf("\n Fils stopp par signal %d\n", WSTOPSIG(stat_files));
        }
        if(WIFCONTINUED(st_files)!=0) printf("Proc. fils continue");
    }
}
```

Code de retour

signal SIGCHLD

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Dans un premier terminal `watch -n 1 'ps -l -forest -C "ex-forksigwait,bash,upstart"'`
- Dans un second terminal `./ex-forksigwait 20 40`
- Dans un troisième terminal `kill pid_fil`
`kill -STOP pid_fil`
`kill -CONT pid_fil`

Sommaire

Création de processus

Création de processus

Introduction
Créat. fork()
Recouv. exec()
CMD : system
Exercice

Terminaison

Caractéristiques
Processus zombies
Code de retour

Pipes

Caractéristiques
Pipes anonymes
Pipes nommés

1 Création de processus

- Introduction
- Créat. fork()
- Recouv. exec()
- CMD : system
- Exercice

2 Terminaison

- Caractéristiques
- Processus zombies
- Code de retour

3 Pipes

- Caractéristiques
- Pipes anonymes
- Pipes nommés

La communication inter-processus

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Il existe plusieurs mécanismes de communication entre processus.
 - Les pipes anonymes : **pipe()**
 - Les pipes nommés : **mkfifo()**
 - Les IPC System V :
 - Les segments de mémoire partagée : shmem (non abordés dans ce cours)
 - Les sémaphores : semop (non abordés dans ce cours)
 - Files de messages : msgget (non abordé dans ce cours)
 - Les sockets : (non abordés dans ce cours)

Caractéristiques

Création de processus

Création de processus

Introduction
Créat. fork()
Recouv. exec()
CMD : system
Exercice

Terminaison

Caractéristiques
Processus zombies
Code de retour

Pipes

Caractéristiques

Pipes anonymes
Pipes nommés

- La communication est unidirectionnelle et faite en mode FIFO (first in first out)
- Ce qui est lu quitte définitivement le tube et ne peut être relu
- La transmission est faite en mode flot continu d'octets
- Pour fonctionner, un tube doit avoir au moins un lecteur et un rédacteur Un tube a une capacité finie
- Une synchronisation de type producteur/consommateur entre lecteurs et rédacteurs:
 - un lecteur peut parfois attendre qu'il y est quelque chose d'écrite avant de lire,
 - un écrivain peut attendre qu'il y ait de la place dans le tube avant de pouvoir y écrire

Types des pipes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Il existe deux types de pipes :

- **Les pipes anonymes :**

- Permettant la communication entre deux processus ayant un ancêtre commun. Ce qui est écrit dans une extrémité du pipe peut être lu de l'autre extrémité.

- **Les pipes nommés :**

- Permettant la communication entre n'importe quels processus en passant par le système de fichier.
- Un pipe nommé est un fichier spécifique de type FIFO. Il peut être ainsi ouvert par plusieurs processus en lecture ou en écriture.
- Lorsque des processus s'échangent des données à travers le pipe nommé, le noyau fait passer les données en interne sans écriture sur le système de fichier.

Les pipes anonymes

Création de
processus

Création de
processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Procédure de fonctionnement :

- Le pipe est créé par l'appel système `pipe()`
- Le processus créateur crée un processus fils par `fork()`
- Le père et le fils choisissent le sens de communication dans le pipe commun
- Ils communiquent par `read()` et `write()` sur le pipe commun, mêmes appels systèmes que sur les fichiers

Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Création d'un pipe :

- Un pipe anonyme est créé avec l'appel système *pipe()* :
`#include <unistd.h>`
`int pipe(int fd[2]);`
- L'appel système *pipe(fd)*, donne un tableau d'entier *fd* de taille 2, crée une paire de descripteurs de fichier, *fd[0]* et *fd[1]*.
- S'il réussit, il renvoie 0, sinon il renvoie -1.
- Le processus peut ensuite écrire à l'extrémité d'écriture, *fd[1]*, à l'aide de l'appel système *write()*, et peut lire à partir de l'extrémité lecture, *fd[0]*, à l'aide de l'appel système *read()*.

Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

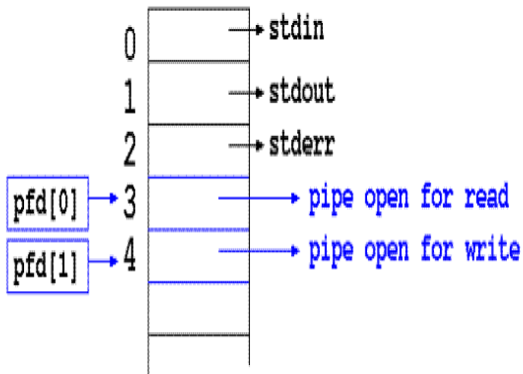
Caractéristiques

Pipes anonymes

Pipes nommés

Descripteurs d'un pipe :

- `int pfd[2]; /* déclaration des descripteurs */`
- `pipe(pfd) ; /* création du pipe ==> deux descripteurs sont alloués dans la table de descripteurs */`



Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Lecture dans un Tube: Primitive read() :

- On peut lire dans un tube à l'aide de la primitive classique read().
- On utilise le descripteur p[0] rendu par pipe(). Dans l'exemple qui suit :

```
char buf[100];
```

```
int p[2];
```

```
...
```

```
read(p[0], buf, 20);
```

- On a une demande de lecture de 20 caractères dans le tube p. Les caractères lus sont rendus disponibles dans la zone buf. Leur nombre est la valeur retour de read().

Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Lecture dans un Tube: Primitive read() :

```
close(p[1]);  
read(p[0], buf, 20);
```

- Une précaution voudrait qu'un processus ferme systématiquement les descripteurs dont il n'a pas besoin: ici on a besoin de lire dans le tube p. On doit fermer le descripteur p[1].
- Cela permet d'éviter des erreurs aboutissant parfois à des situations d'interblocage (deadlock): des processus communiquent, mais chacun attend que l'autre commence.

Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Écriture dans un Tube: Primitive write():

- On peut écrire dans un tube avec la primitive classique write(). L'écriture est faite en utilisant le descripteur p[1] cette fois-ci. La séquence qui suit:

```
char buf[100];
```

```
int p[2];
```

```
...
```

```
close p[0];
```

```
buf = "texte à écrire ";
```

```
write(p[1], buf, 20);
```

est une demande d'écriture de 20 caractères dans le tube de descripteur ouvert p[1].

Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

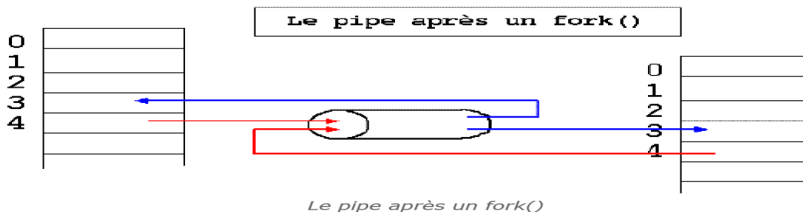
Écriture dans un Tube: Primitive write():

- La séquence à écrire est prise dans la zone buf.
- La valeur retour de write() est le nombre d'octets ainsi écrits. Là aussi, on a fermé le descripteur p[0] de lecture.
- L'écriture dans un tube est atomique (tout est écrit ou rien n'est écrit) et n'interfère pas avec d'autres écrivains éventuels.
- Les 20 caractères écrits ici seront consécutifs dans le tube.

Les pipes anonymes

Héritage du pipe :

- Après le `fork()` le pipe est dupliqué dans le processus fils puisque la table des descripteurs fait partie du PCB du processus père.



- Il y'aura deux accès dans chaque sens (un accès pour le père et un autre pour le fils). Il faudra que chaque processus choisisse le sens d'utilisation.

Les pipes anonymes

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Risque de blocage :

- La coordination entre le processus lecteur et le processus rédacteur doit être prévue, sinon il y'a risque de blocage de l'un des processus :
 - buffer plein ==> write(p[1], ...) bloque
 - buffer vide ==> read(p[0], ...) bloque
- Par contre, p[1] closed et buffer vide ==> read(p[0], ...) returns 0 (end of file)
- Les lectures / écritures peuvent être rendues non-bloquantes en utilisant l'option O_NODELAY (System V), ou O_NONBLOCK (POSIX) la primitive

```
status = fcntl(pfd[0], G_GETFL) ;  
fcntl(pfd[0], F_SETFL, status | O_NONBLOCK) ;
```

Les pipes anonymes

Création de processus

Création de processus

- Introduction
- Créat. fork()
- Recouv. exec()
- CMD : system
- Exercice
- Terminaison
- Caractéristiques
- Processus zombies
- Code de retour

Pipes

- Caractéristiques
- Pipes anonymes**
- Pipes nommés

Exemple

```
#include <unistd.h>
int main()
{
    int pid, pip[2];
    char instring[20];
    pipe(pip);
    pid = fork();
    if (pid == 0) /* fils: envoie un message a son parent*/
    {
        write(pip[1], "Salut !", 7); /* envoyer un message */
    }
    else /* parent : recoit le message envoy par le fils */
        read(pip[0], instring, 7); /* lire partir du pipe*/
}
```


Les pipes nommés

Création de processus

Création de processus

Introduction
Créat. fork()
Recouv. exec()
CMD : system
Exercice

Terminaison

Caractéristiques
Processus zombies
Code de retour

Pipes

Caractéristiques
Pipes anonymes
Pipes nommés

- Les pipes sans nom sont un mécanisme élégant, cependant, ils ont plusieurs inconvénients :
 - Ils ne peuvent être partagés que par des processus avec un ancêtre commun
 - En outre, ils cessent d'exister dès que les processus qui les utilisent se terminent, de sorte qu'ils doivent être recréés chaque fois qu'ils sont nécessaires
 - C'est pas pratique pour les applications client-serveur

Les pipes nommés

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Un pipe nommé est un fichier spécifique de type FIFO
- Il est similaire à un pipe anonyme sauf qu'il fait partie du système de fichier
- Il peut être ainsi ouvert par plusieurs processus en lecture ou en écriture
- Lorsque des processus s'échangent des données à travers le pipe nommé, le noyau fait passer les données en interne sans écriture sur le système de fichier

Les pipes nommés

Création

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombis

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- Par les commandes shell UNIX, par exemple:
`mknod <filename> p`
`mkfifo a=rw <filename>`
- Par appels système :
`mknod(char *pathname, mode_t mode, dev_t dev);`
Exemple : `mknod("/tmp/myfifo", S_IFIFO | 0660, 0);`
- Si `pathname` existe déjà, l'appel échoue avec l'erreur *EEXIST*.
La commande `ls -l` par exemple, les affichent avec la lettre *p*, comme pipe, en position type de fichier.

Les pipes nommés

Création

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

- On peut afficher les attributs du tube créé comme on le fait avec les fichiers :
hlakhlef > ls -l tube
prw-r--r-- 1 hlakhlef hlakhlef 0 Mar 2 11:36 tube|
- De la même façon, il est possible de modifier des permissions d'accès :
hlakhlef> chmod g+rw tube
hlakhlef > ls -l tube
prw-rw-r-- 1 hlakhlef hlakhlef 0 Mar 2 11:36 tube|

Les pipes nommés

exemple (client-serveur)

Création de processus

Création de processus

Introduction

Créat. fork()

Recouv. exec()

CMD : system

Exercice

Terminaison

Caractéristiques

Processus zombies

Code de retour

Pipes

Caractéristiques

Pipes anonymes

Pipes nommés

Serveur.c

```
#include <fcntl.h>
...
#define PIPE "fifo"
int main() {
    int fd; char readbuf[20];
    mknod(PIPE, S_IFIFO | 0660, 0); // cration du pipe
    fd= open(PIPE, O_RDONLY, 0); // ouvrir le pipe
    for (;;) {
        if (read(fd, &readbuf, sizeof(readbuf)) < 0){ // lire du pipe
            perror("Error reading pipe"); exit(1);
        }
        printf("Received string: %s\n", readbuf);
    }
    exit(0);
}
```

Les pipes nommés

exemple (client-serveur)

Création de
processus

Création de
processus

Introduction
Créat. fork()
Recouv. exec()
CMD : system
Exercice

Terminaison

Caractéristiques
Processus zombies
Code de retour

Pipes

Caractéristiques
Pipes anonymes
Pipes nommés

Client.c

```
#include <stdio.h>

...

#define PIPE fifo

int main(){
    int fd;
    char writebuf[20] = Hello; // ouvrir le pipe
    fd= open(PIPE, O_WRONLY, 0); // créer dans le pipe
    write(fd, writebuf, sizeof(writebuf));
    exit(0);
}
```