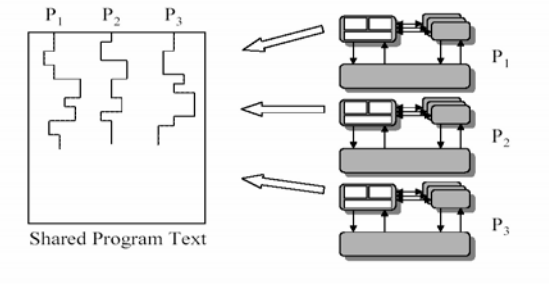
Processos em UNIX

Sistemas Operativos, 2004-2005

Processes sharing a program



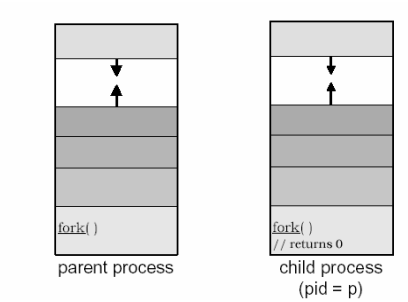
Running Unix commands from C

```
main() {
   printf(”Files in Directory are: \n");
   system(”ls -l");
}
```


Unix Process

- Each process has its own address space subdivided into: **text, data, & stack segment**
- ***Process identifier*** **(PID):** User handle for the process (descriptor)
- **int getpid()** will return the pid of the process.
- Try also: **ps** and **ps -aux**

Creating a Process



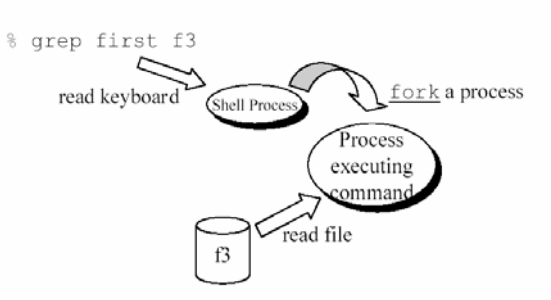
5

Creating/Destroying Processes

- UNIX **fork** creates a process:
  - Creates a new address space
  - Copies text, data, & stack into new address space
- Provides child with access to open files.
- UNIX **wait** allows a parent to wait for a child to terminate
- UNIX **exec** allows a child to run a new program

6

Executing a Unix Command



7

Unix fork

int fork();

DESCRIÇÃO:

***fork()* cria um novo processo (filho) absolutamente idêntico ao processo pai. O processo filho herda todo o contexto do processo pai e continua a executar o mesmo código na instrução seguinte ao *fork*. Isto significa que o processo filho partilha os ficheiros que foram anteriormente abertos, a directoria corrente, os dispositivos de I/O, o nível de prioridade, etc). No entanto, o processo filho passa a ter um novo *pid*.**

VALORES DE RETORNO:

success ---------- fork() devolve 0 ao processo filho,
e o pid (>0) do filho ao processo pai
error ---------- fork() devolve -1 ao processo pai.

8

Unix: wait and exit

int wait(int *status);

O processo pai pode bloquear à espera da terminação do processo filho. A função *wait()* devolve o pid do processo que terminou.

exit(int code);

Termina um processo e devolve um código ao processo pai.

9

Creating a Unix Process

```
int main()
{
    int  id_proc,status;
    id_proc = fork();
    if(id_proc = = 0){
          printf("Olá, eu sou o processo filho (%d)!\n",id_proc);
          sleep(5);
          exit(1);
          }
    else if(id_proc > 0){
          printf("Oi, eu sou o processo pai, e criei um
                                  processo filho com pid:%d \n",id_proc);
          wait(&status);
          exit(1);
          }
    else{
          printf("Erro no fork !!! \n");
          exit(-1);
          }
}
```

Concurrent Processes

```
#include <stdio.h>
#define SIZE 9
int main() {
int pid, i, Answer;
int Numbers[SIZE]={1,2,3,4,5,6,7,8,9};
int x=100;
pid = fork();
if ( pid == 0 ) /* child code begin */ {
      Answer = 0;
      x=200;
      for (i = 0 ; i < SIZE ; i++) {
            Answer = Answer + Numbers[i];
          }
      printf("Child: sum = %d\n", Answer);
      _exit(0);
      } /* child code end */
if ( pid < 0 ) {
      fprintf(stderr, "fork failed\n");
      exit(1); }
Answer = 1;
x=300;
for (i = 0 ; i < SIZE ; i++) {
      Answer = Answer * Numbers[i];
      }
      printf("Parent: product = %d\n", Answer);
}
```

Get process identifiers

int getpid()
int getppid()

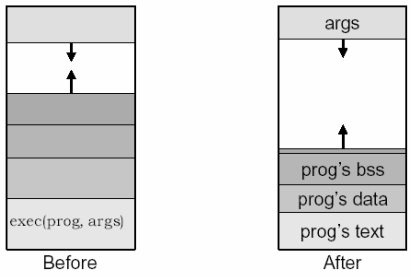
DESCRIÇÃO:

getpid() devolve o pid do próprio processo.

getppid() devolve o pid do seu processo pai.

12

exec(): loading a new program



13

Unix: exec()

int exec("ficheiro",arg0,arg1,...);

A primitiva *exec()* modifica o segmento de dados e texto do processo. O contexto núcleo e a pilha mantém-se os mesmos.

A partir da chamada a *exec()* o programa passa a ser outro.

14

Example: fork/exec

```
if (fork( ) == 0) {
    // child process
    — set up I/O in child —
    execv(newprogram, parameters);
    // load new image
    // if we get here, there's a problem
}
// parent process continues here
```

15

Example

```
int main()
{
    int  id_proc,status,id_term;
    id_proc = fork();
    if(id_proc = = 0){
            printf("Olá, eu sou o proc. filho(pid=%d),getpid());
            sleep(5);
            execl("ls","ls",0);   /* executa comando ls da shell.... */
            printf("Viva o Benfica!.....\n");
            exit(1);
            }
    else if(id_proc > 0){
            printf("Oi, eu sou o processo pai (pid=%d),
                    e criei um processo filho um filho com pid:%d PPID=%d \n",
                                            getpid(),id_proc, getppid());
            id_term=wait(&status);
            print("após terminacao do pid:%d\n",id_term);
            exit(1);
            }
    else{
            printf("Erro no fork !!! \n");
            exit(-1);
            }
}
```

Another Example

```
#include       <sys/wait.h>
#define NULL    0
int main (void)
{
   if (fork() == 0){   /* código processo filho */
      execve("child","ls",NULL);
      printf("O BENFICA é o MAIOR ! \n");
      exit(0);
    }
/* código processo pai */
   printf("Process[%d]: processo pai ...\n", getpid());
   sleep(2);
   if(wait(NULL) > 0)
            printf("Process[%d]: detectou terminação do filho \n",
                     getpid());
   printf("Process[%d]: a terminar ...\n", getpid());
}
```

17

Child Process

```
int main (void)
{
/* The child process's new program
   This program replaces the parent's program */
    printf("Process[%d]: child in execution ...\n",
          getpid());
    sleep(1);
    printf("Process[%d]: child terminating ...\n",
          getpid());
}
```

18

Quantas vezes aparece o printf?

```
int x=10;
fork();
fork();
fork();
fork();
fork();
printf("x = %d \n",x);
```

19

QUIZ#1: Como criar 60 processos?

```
....
.int i,status;
.for(i=0; i <60; i++){
      if(fork() == 0){ // processo filho...
            rotina(i);
            }
      else{ // pai...
            }
      }
Printf("pai.... Waiting....\n");
Wait(&status);
....
....
....
....
....
void rotina(int i){
printf("id = %d \n",i);
....
exit(0);
}
```