

Processos no UNIX

Walter Fetter Lages

`w.fetter@ieee.org`

Universidade Federal do Rio Grande do Sul

Escola de Engenharia

Departamento de Engenharia Elétrica

Programa de Pós-Graduação em Engenharia Elétrica

ELE213 Programação de Sistemas de Tempo Real



Introdução

MULTICS Multiplexed Information and Computing Service

UNICS Uniplexed Information and Computing Service

UNIX

- System V
- BSD=Berkeley Software Distribution
- POSIX=Portable Operating System Interface
- IEEE 1003.x
- IEEE 1003.4 Real-time Extension



Criação de Processos

- `pid_t fork(void)`
 - Chamada de sistema clone
 - Processo pai
 - Processo filho
 - Process identifier (pid)
 - Fork retorna 0 para o filho
 - Fork retorna pid do filho para o pai
 - Descritores e privilégios do pai são herdados pelo filho
 - Locks e sinais não são herdados
 - `vfork()` -> não duplica todo o espaço de endereços

Exemplo

```
pid=fork();  
if(pid < 0)  
{  
    /* falha do fork */  
}  
else if (pid > 0)  
{  
    /* codigo do pai */  
}  
else  
{  
    /* codigo do filho */  
}
```



Identificação de Processos

- pid

```
pid_t  getpid(void)
pid_t  getppid(void)
```

- User id

```
uid_t  getuid(void)
uid_t  geteuid(void)
```

- Group id

```
gid_t  getgid(void)
gid_t  getegid(void)
```

Notificação de Término

- Sinal SIGCHLD enviado para o pai
 - Notificação assíncrona
- `pid_t wait(int *staloc)`
staloc ponteiro onde é armazenado o status retornado pelo filho ao terminar

```
pid_t waitpid(pid_t pid, int *staloc,  
              int options)
```

`pid == -1` espera qualquer filho

`pid > 0` espera pelo filho com `id == pid`

`pid == 0` espera por filho com `gid == pid`

`pid < -1` espera por filho com `gid == abs(pid)`



Notificação de Término

```
pid_t waitpid(pid_t pid, int *staleoc,  
              int options)
```

options bitwise OR

WNOHANG Retorna 0 ao invés de bloquear

WUNTRACED Se a implementação suporta job control, retorna o status de término de qualquer filho especificado por pid.



Funções exec

```
int execl(const char *pathname,  
          const char *arg0, ...)  
int execv(const char *pathname,  
          char *const argv[])  
int execlp(const char *pathname,  
            const char *arg0, ...,  
            char *const envp[])  
int execve(const char *pathname,  
            char *const argv[],  
            char *const envp[])  
int execlp(const char *filename,  
            const char *arg0, ...)  
int execvp(const char *filename,  
            char *const argv[])
```




Troca de UID e GID

- UID

```
int setuid(uid_t uid)
```

```
int setreuid(uid_t ruid, uid_t euid)
```

```
int seteuid(uid_t uid)
```

- GID

```
int setgid(gid_t gid)
```

```
int setregid(gid_t rgid, gid_t egid)
```

```
int setegid(gid_t gid)
```

Grupos de Processos

- Cada processo pertence a um grupo de processo e tem um `pgid`
- Cada grupo de processo tem um líder
 - Para o líder, `pgid=pid`

```
pid_t getpgrp(void)
```

```
int setpgid(pid_t pid, pid_t pgid)
```

- Um processo só pode setar o `pgid` de si próprio ou dos seus filhos

Sessões

- Uma sessão é um conjunto de grupos de processos
- `pid_t setsid(void)`
 - O processo torna-se um líder de uma nova sessão e líder de um novo grupo
 - Se o processo já for líder de grupo, a chamada retorna erro
 - Cada sessão pode ter um único terminal controlador

Processos, Grupos e Sessões

