GSI018 – Sistemas Operacionais

Gerência de Processos

Universidade Federal de Uberlândia
Faculdade de Computação
Prof. Dr. rer. nat. Daniel D. Abdala

Na Aula Anterior...

- Interrupções por software e hardware;
- Temporizadores;
  - RTC;
  - Temporizadores via software.

Nesta Aula

- O Conceito de Processo;
  - Relação com Multiprogramação e Timesharing;
- Estrutura de um Processo;
- Estados de um Processo;
- Hierarquia de Processos;
- A Especificação POSIX;
- Criação e Término de Processos.

Processos

- "Um Programa em Execução";
  - "Um Programa Apto a ser Executado!";
- Programa → "Receita" | Processo → "Bolo";
- Um processo é a abstração de um programa sob o ponto de vista do sistema operacional;
- Uma das abstrações fundamentais em SOs;

Relação Processos ↔ Multiprogramação

- Em um sistema monotarefa há pouca ou nenhuma necessidade de um modelo complexo de processos;
- Em sistemas multitarefas o conceito de processos é fundamental;
- Ilusão de que todos os recursos do sistema estão disponíveis para o processo;
  - Na realidade o processador está disponível apenas uma parcela (quantum) de tempo;
  - Apenas uma parcela da memória está disponível;
  - Apenas os dispositivos de E/S que o processo requisitou e não já estavam em utilização;
  - Apenas os arquivos ainda não utilizados por outros processos;

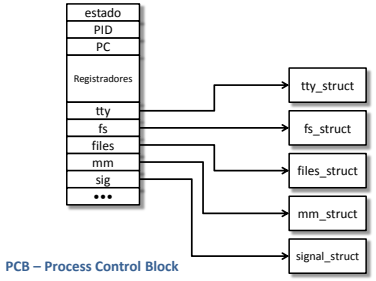
Relação Processos ↔ Multiprogramação

- Multiprogramação/Multitarefa induz a "ilusão de paralelismo";
- O processo é a peça fundamental neste esquema;
- Cada processo recebe uma "fatia" de tempo de atenção da UCP;
- Como a fatia de tempo é muito curta apenas alguns milisegundos, o usuário tem a impressão de que os programas estão sendo executados em paralelo;
- Trocar o programa que recebe atenção da UCP requer que o SO saiba onde os processos foram interrompidos;
- Também é necessário registrar os valores do PC, dos registradores, descritores de arquivos, terminal associado, espaço de endereços, etc.

Descritores de Processos

- Também chamado de PCB – Process Control Block (Bloco de Controle de Processo);
- Contém toda a informação necessária para:
  - Agendar a execução do processo;
  - Colocar um processo em espera;
  - Retomar a execução de um processo;
- Provavelmente a estrutura de dados mais complexa do SO;
  - Referências para diversas outras estruturas;

7



PCB – Process Control Block

8

Estados de um Processo

- Processos podem estar em diversos estados:
  - Dormindo
  - Rodando
  - Dormindo não Interrompível
  - Parado
  - Zumbi

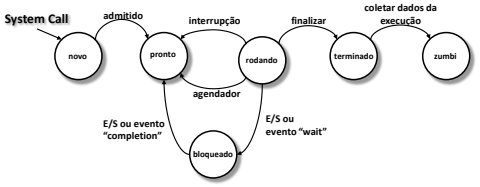
9

Estados de Um Processo

Estado	Flag	Descrição
Dormindo	S	Usualmente esperando que um evento ocorra tal como um sinal ou que entrada de dados esteja disponível
Rodando	R	Apto a ser executado. Pode tanto estar sendo executado ou na fila de agendamento de processos
Dormindo não interrompível	D	(esperando) usualmente, esperando E/S completar
Parado	T	Usualmente parado pelo shell ou sob o controle de um depurador
Zumbi	Z	Processo terminou e está esperando um "wait" do processo pai
	N	Tarefa de baixa prioridade "nice" (uso de `<<renice>>`)
	s	Processo é um líder de sessão
	+	Processo está no grupo de processos do plano de frente
	l	Processo possui múltiplos treads
	<	Processo de alta prioridade

10

Estados de um processo



11

A Especificação POSIX

- **POSIX – Portable Opeating System Interace**
- Padronização feita pela IEEE para principalmente nome e função de chamadas de sistema;
- A maioria dos sistemas Baseados em UNIX hoje em dia são compatíveis com POSIX;
- O Objetivo é facilitar a portabilidade de aplicações entre sistemas baseados no UNIX.

12

Criação de Processos

- Em geral, todo processo é iniciado por outro processo;
- Processos só podem ser iniciados pelo sistema operacional, consequentemente, para criar um novo processo uma **Syscall** é requerida;
- As Syscalls previstas no padrão POSIX relacionadas a criação de processos são:
  - system
  - exec (e suas variantes)
  - fork
  - wait

13

Criação de Processos

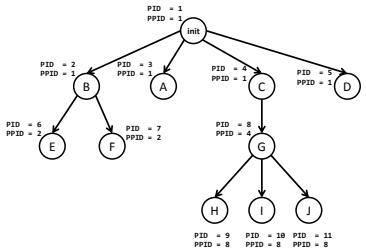
- Quando um processo é criado, apenas o seu descritor de processo (PCB) é criado;
- O processo criado é uma cópia exata do processo que o criou;
- Seu segmento de dados, texto, pilha, PC, demais registradores, descritores de arquivo e espaço de endereçamento são exatamente o mesmo;
- Tal fato torna a criação de um novo processo eficiente e rápida;
- Quando o novo processo executa, as coisas podem começar a variar;
- Novos arquivos podem ser abertos, o endereço no PC variará em relação ao processo pai, etc.

14

Hierarquia de Processos

- Em geral, todo processo é iniciado por outro processo;
- O **processo A** que cria um novo **processo B** é chamado de **pai de B**;
- O **processo B** é chamado de **filho de A**;
- Existem basicamente dois tipos de processos:
  - Processos do Sistema → parte do sistema operacional; Em geral rodam em **modo kernel**;
  - Processos do Usuário → softwares aplicativos e demais processos; rodam em **modo usuário**;
- No Linux apenas um processo não possui pai (**init**) que é o processo inicial do sistema operacional;

15



Exemplo de Hierarquia de Processos

16

Criação de Processos (**system**)

- Usado para executar um programa de dentro de outro programa;
- A chamada do sistema <<system>> permite que um novo processo seja criado;
- Um terminal virtual é criado e o programa especificado como argumento é executado como um novo processo;

```
#include <stdlib.h>

int system (const char *string);
```

```
//equivalente a :

$sh -c string
```

17

Criação de Processos (**exec**)

- A chamada do sistema exec* (variantes) permite que os seguimentos de texto e dados estático de um processo seja substituída;
- As variantes permitem controle fino dos argumentos a serem passados e da localização do arquivo executável que substituirá o código do processo;
- Geralmente um processo é criado usando <<fork>> e então substituído usando <<exec*>>.

```
#include <unistd.h>

char **environ;

int execl(const char *path, const char *arg0, ..., (char *)0);
int execlp(const char *file, const char *arg0, ..., (char *)0);
int execle(const char *path, const char *arg0, ..., (char *)0, char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
```

18

Criação de Processos (**fork**)

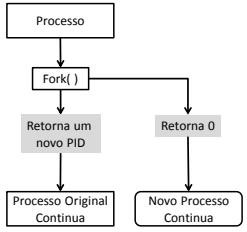
- Provavelmente a chamada do sistema mais utilizada para criação de novos processos;
- Ela bifurca a execução do código, prosseguindo a partir do mesmo ponto em ambos os processos (pai, e filho);

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

pid_t fork(void);
```

19

Criação de Processos



20

Espera por Termino de um Processo

- Em alguns casos, pode ser desejável que o processo pai espere que o processo filho termine sua execução antes que ele possa continuar;
- A chamada do sistema <<`wait`>> coloca o processo que a chamou em estado de espera até que um de seus processos filhos termine;
- Ela retorna o PDI do processo que terminou e como argumento o estado do processo que terminou;

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

21

Espera por Termino de um Processo

- O ponteiro <<`stat_val`>> aponta para uma estrutura de dados que informa acerca do estado do processo;
- Ele pode ser interpretado facilmente utilizando macros definidas no header sys/wait.h;

Macro (sys/wait.h)	Descrião
WIFEXITED(stat_val)	Retorna ≠ 0 se a proc. Filho terminou normalmente
WEXITSTATUS(stat_val)	Se WIFEXITED ≠ 0, retorna o código de saída do proc. filho
WIFSIGNALED(stat_val)	≠ 0 se o proc. filho terminou em um sinal não tratado
WTERMSIG(stat_val)	Se WIFSIGNALED ≠ 0, retorna o número do sinal
WIFSTOPPED(stat_val)	≠ 0 se o proc. filho foi interrompido
WSTOPSIG(stat_val)	Se WIFSTOPPED ≠ 0, retorna o número do sinal

22

Término de Processos

- Um processo pode ser terminado a qualquer momento por meio da invocação da chamada do sistema <<`exit`>>;

```
#include <stdlib.h>

void exit(int status);
```

23

Processos no Windows

- Processos no Windows não seguem as mesmas convenções prevista no padrão POSIX;
- Em geral a função <<`CreateProcess`>> faz o trabalho de ambos <<`fork`>> e <<`exec*`>>;

24

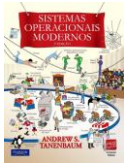
Comparativo UNIX*/Win32 API

Funções relacionadas a processos

UNIX	Win32	Descrição
fork	CreateProcess	Cria um novo processo
waitpid	WaitForSingleObject	Espera que um processo termine
execve	-	Substitui a imagem de um processo
exit	ExitProcess	Conclui a execução

25

Bibliografia - Básica



- 3ª Edição
- Páginas 50-56

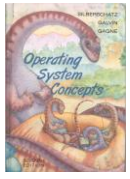
26

Bibliografia - Adicional

- 4ª Edição
- Páginas 461-493

27

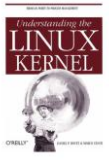
Bibliografia - Básica



- 7ª edição
- Páginas 80-90

28

Bibliografia - Adicional



- Páginas 64 a 95

29