FATEC

CENTRO PAULA SOUZA
COMPETÊNCIA EM EDUCAÇÃO PÚBLICA PROFISSIONAL
Faculdade de Tecnologia Prof. Jessen Vidal
São José dos Campos

GOVERNO DO ESTADO DE SÃO PAULO

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Disciplina: Sistemas Operacionais I

Aula 07: Processos P2

Prof. Diogo Branquinho Ramos
diogo.branquinho@fatec.sp.gov.br
São José dos Campos - SP

Roteiro

- Visualização genérica de escalonamento de processos
- Criação de Processos
- Término de processos

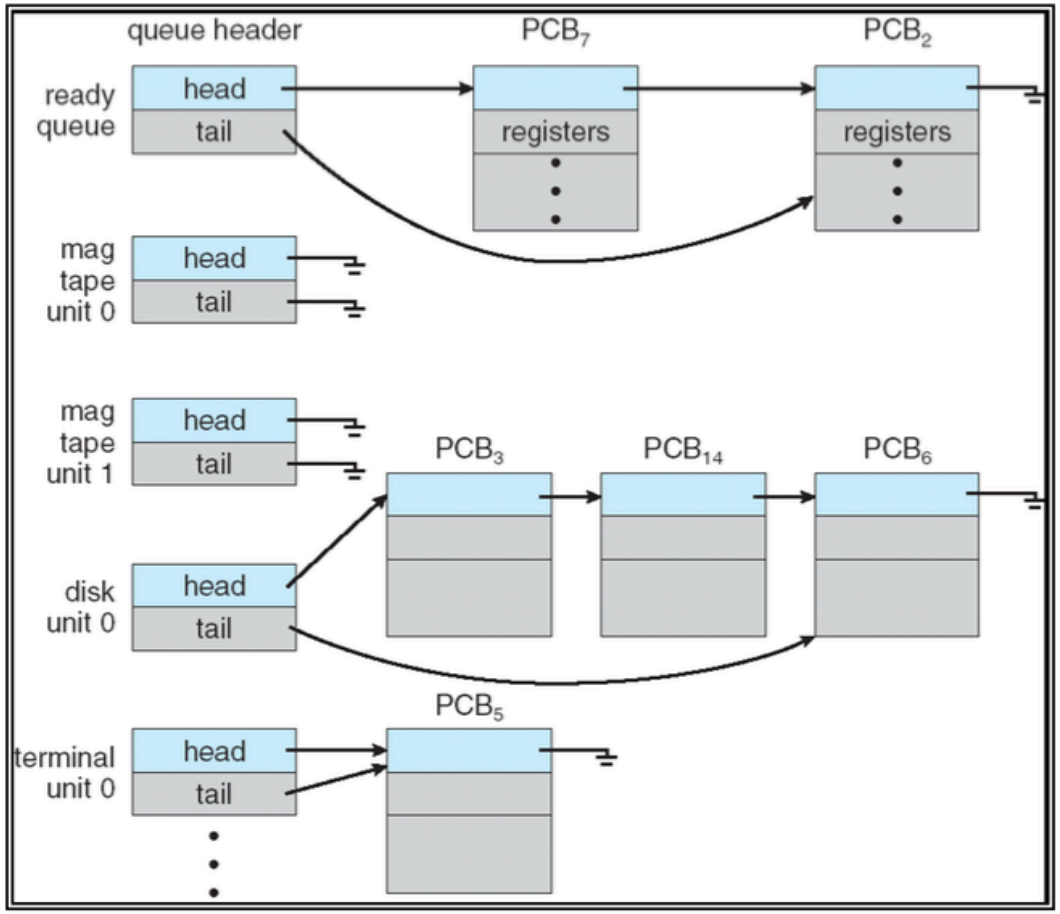
Escalonamento de processo

- **Qual o objetivo da multiprogramação?**
  - Ter mais de um processo na RAM!
- **Qual o objetivo do compartilhamento de tempo?**
  - Alternar a CPU entre os processos com muita frequência, para que os usuários interajam com cada programa em execução.
- **Resultado**
  - Em um sistema com um único processador nunca haverá mais do que um processo em execução, mas a sensação do usuário é que tem vários processos executando simultaneamente!

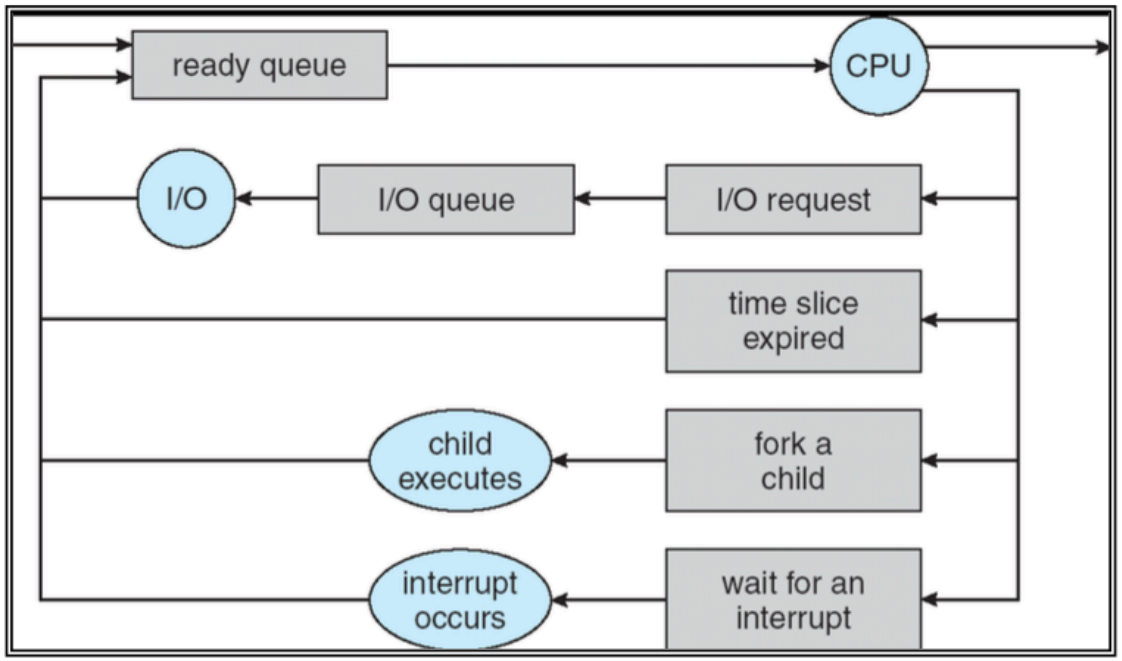
Filas de escalonamento de processo

- **Fila de tarefas**
  - Conjunto de todos os processos no sistema.
- **Fila de prontos**
  - Conjunto de todos os processos residindo na memória principal, prontos e esperando para execução.
- **Filas de dispositivo**
  - Conjunto de processos esperando por um dispositivo de E/S.

Possíveis filas no sistema



Representação do escalonamento de processos



Escalonadores

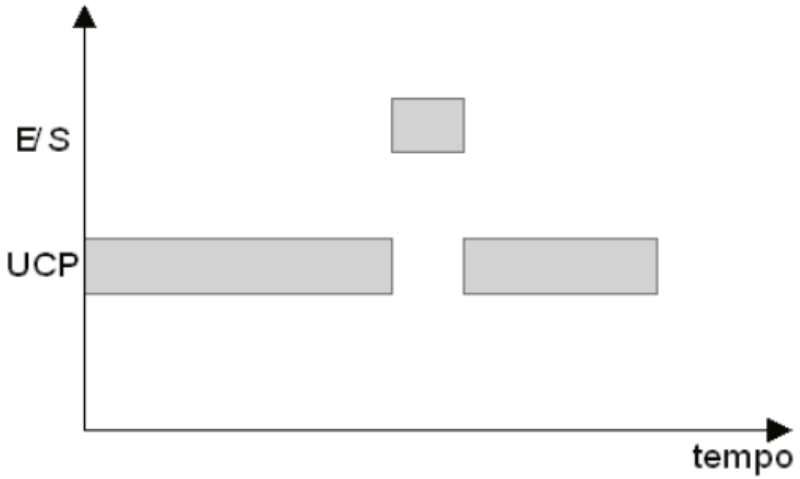
- **Processos estão migrando entre as diversas filas!**
- **Escalonador a longo prazo (ou escalonador de tarefas)**
  - Seleciona quais processos devem ser trazidos para a fila de prontos.
  - Invocado com baixa frequência → lento (s, min).
- **Escalonador a curto prazo (ou escalonador de CPU)**
  - Seleciona qual processo deve ser executado em seguida e aloca CPU.
  - Invocado com muita frequência → rápido (ms).

Escalonadores

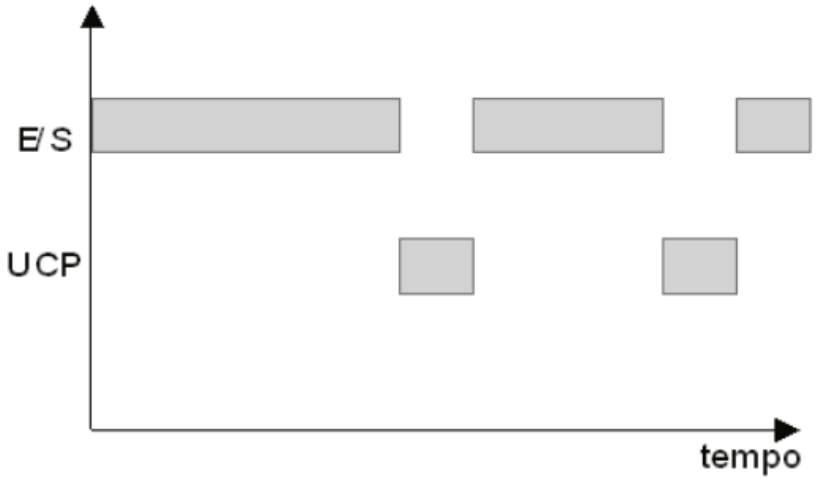
Escalonador a longo prazo

- **Controla o grau de multiprogramação estável**
  - A taxa média de criação do processo é igual a taxa de saída média dos processos que deixam o sistema.
- **Mais tempo para decidir**
- **Tipos de Processo**
  - I/O-bound
    - Processo que gasta mais tempo com E/S.
  - CPU-bound
    - Processo que gasta mais tempo realizando cálculos.

CPU-bound x I/O-bound



(a) CPU-bound



(b) I/O-bound

Escalonadores

Escalonador a longo prazo

- **O importante é o equilíbrio!**
  - Se todos os processos forem *I/O-bound*
    - Fila de prontos vazia.
    - Escalonador de curto prazo ocioso.
  - Se todos os processos forem *CPU-bound*
    - Fila de espera de E/S vazia.
    - Dispositivos pouco usados.

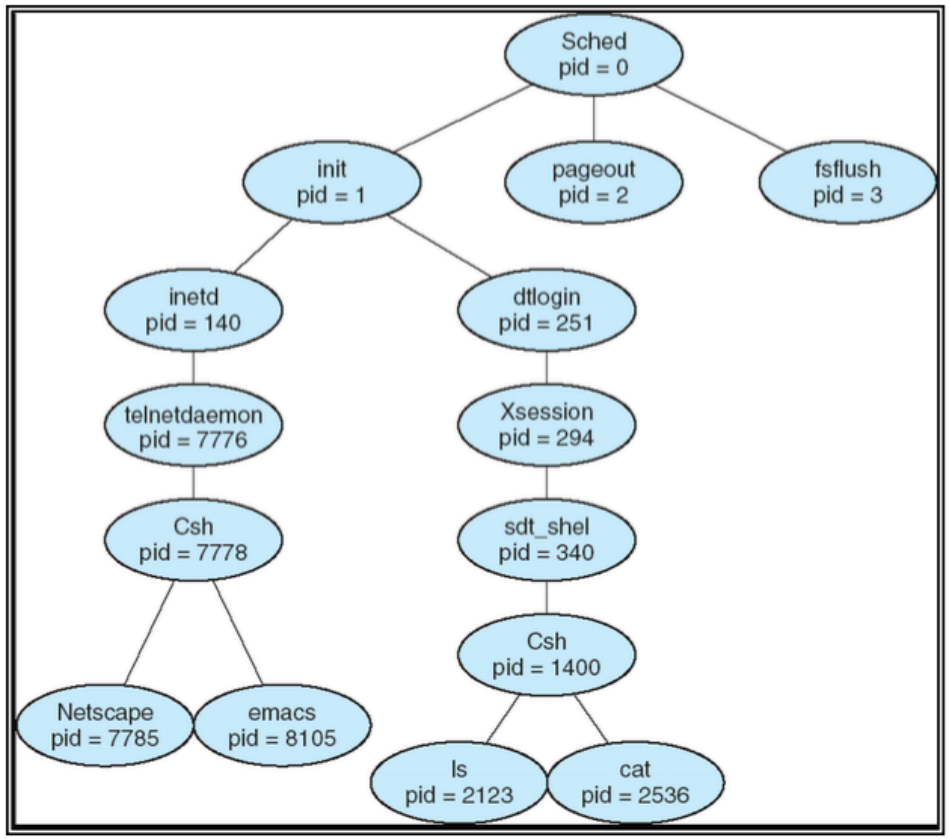
Escalonamento de meio termo

- **Objetivo**
  - Reduzir o grau de multiprogramação
    - Pode ser vantajoso remover processos da memória e da disputa pela CPU, que podem ser reintroduzidos depois.
    - Esse esquema é chamado de swapping.

Criação de processo

- **Árvore de processos**
  - Processo pai cria processos filhos que, por sua vez, criam outros processos (*syscall* "criar processo").
  - PID: valor inteiro maior ou igual a zero e exclusivo.
  - Solaris
    - sched: processo-pai.
    - pageout e fsflush: gerência de memória e de sist. de arquivos.
    - init: processo-pai dos processos do usuário.
    - inetd: processos de rede.
    - dtlogin: *login* do usuário.
    - ...

Árvores de processos em um Solaris típico



Criação de processo

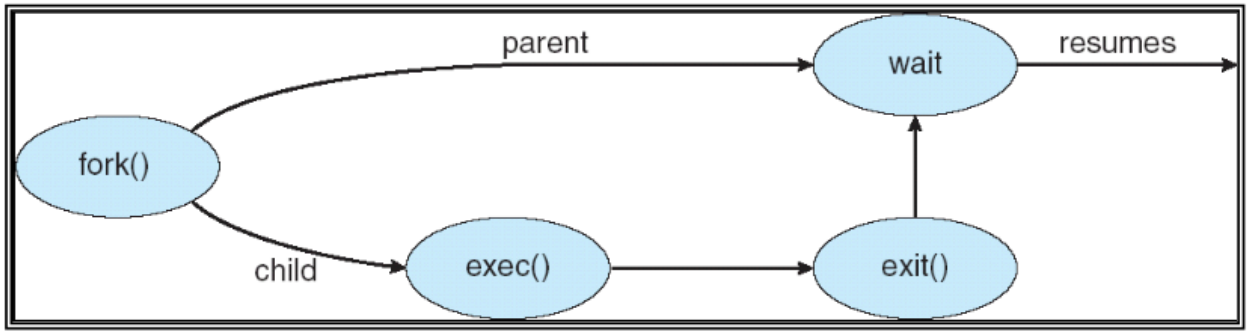
- **Compartilhamento de recursos**
  - P-pai e p-filhos compartilham todos os recursos; ou
  - Filhos compartilham subconjunto dos recursos do pai; ou
  - Pai e filho não compartilham recursos (p-filhos interagem diretamente com o SO).
- **P-pai pode passar dados de inicialização.**
  - Processo para exibir uma imagem na tela → entrada: nome do arquivo.
- **Execução**
  - Pai e filhos executam concorrentemente; ou
  - Pai espera até que filhos terminem.

Criação de processo

- **Espaço de endereços**
  - Filho duplicata do pai; ou
  - Filho tem um programa carregado.
- **Exemplos do UNIX**
  - Syscall ***fork*()** cria novo processo (cópia do espaço de endereços do processo original).
    - Retorno do *fork*() é 0 para o p-filho e diferente de zero para o p-pai.
    - Um p-pai possui limites de criação de p-filhos.

Criação de processo

- **Exemplos do UNIX**
  - *Syscall* ***exec()*** usada após um ***fork()*** para substituir o espaço de memória do processo por um novo programa.
    - →Os dois processos podem se comunicar e seguir seus caminhos separados.
    - →O pai pode criar mais filhos, ou pode emitir uma *syscall wait()* para sair da fila de prontos até que a execução do filho termine.



Criação de processo no POSIX

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

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
       fprintf(stderr, "Fork Failed");
       exit(-1);
    }
    else if (pid == 0) { /* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else { /* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
       exit(0);
    }
}
```

pid é só o nome da variável!

Cuidado para não confundir!

Veja: http://www.dca.ufrn.br/~adelardo/cursos/DCA409/node34.html

```
#include <stdio.h> #include <sys/types.h> #include <unistd.h>
main()
{
  int pid;

  pid=fork();

  if ( pid < 0 ) {  fprintf(stderr,"erro\n"); exit(1);  }

  if ( 0 ==pid )
    printf("FILHO: \t id is %d, pid (valor)is %d\n",getpid(), pid);
  else
     printf("PAI: \t id is %d, pid (filho)is %d\n",  getpid(), pid);
   /* este comando executado duas vezes..*/
   system("date");
}
```

pid é só o nome da variável!

```
alunos:~/so/cprogs/forks crocker$ ./fork1x
PAI:      id is 22571, pid (filho)  is 22572
FILHO:  id is 22572, pid (valor) is 0
Tue Mar 29 12:19:44 WEST 2005
Tue Mar 29 12:19:44 WEST 2005
```

Criação de processo no Win32

```
#include <stdio.h>
#include <windows.h>

int main(VOID)
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

    // allocate memory
    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // create child process
    if (!CreateProcess(NULL, // use command line
     "C:\\WINDOWS\\system32\\mspaint.exe", // command line
     NULL, // don't inherit process handle
     NULL, // don't inherit thread handle
     FALSE, // disable handle inheritance
     0, // no creation flags
     NULL, // use parent's environment block
     NULL, // use parent's existing directory
     &si,
     &pi))
    {
       fprintf(stderr, "Create Process Failed");
       return -1;
    }
    // parent will wait for the child to complete
    WaitForSingleObject(pi.hProcess, INFINITE);
    printf("Child Complete");

    // close handles
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

Criação de processo em Java

```
import java.io.*;

public class OSProcess
{
 public static void main(String[] args) throws IOException {
  if (args.length != 1) {
   System.err.println("Usage: java OSProcess <command>");
   System.exit(0);
  }

  // args[0] is the command
  ProcessBuilder pb = new ProcessBuilder(args[0]);
  Process proc = pb.start();

  // obtain the input stream
  InputStream is = proc.getInputStream();
  InputStreamReader isr = new InputStreamReader(is);
  BufferedReader br = new BufferedReader(isr);

  // read what is returned by the command
  String line;
  while ( (line = br.readLine()) != null)
    System.out.println(line);

  br.close();
 }
}
```

Término de processo

- **Processo executa última instrução e "pede" ao sistema operacional para excluí-lo (*exit*).**
  - Dados de saída do filho para o pai (via ***wait***);
  - Recursos do processo são desalocados pelo sistema operacional.
- **Pai pode terminar a execução dos processos dos filhos (*abort*):**
  - Filho excedeu recursos alocados (é pr ioná-los);
  - Tarefa atribuída ao filho não é mais ne
  - Se o pai estiver saindo.
    - Alguns SOs não permitem que o filho continue se o pai terminar: *término em cascata* → conduzido pelo SO.