

Sistemas Operacionais

Introdução a Processos

Prof. Robson Siscoutto

e-mail: robson@unoeste.br

Sistemas Operacionais

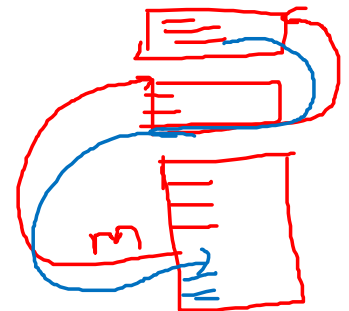
Introdução a Processos

- Introdução;
- Modelo do processo - Ambiente;
- Estados e mudanças do processo (Ciclo de Vida);
- Tipos de processos;
- Escalonamento de Processos
- Comunicação entre processos;
- Processos no Linux - Controle de Processos no Linux
- Concorrência entre Processos
 - Problemas de sincronização;
 - Soluções de hardware e software;
- Escalonamento não Peemptivo e Preemptivo;
- Deadlock
- Threads

Introdução a Processos

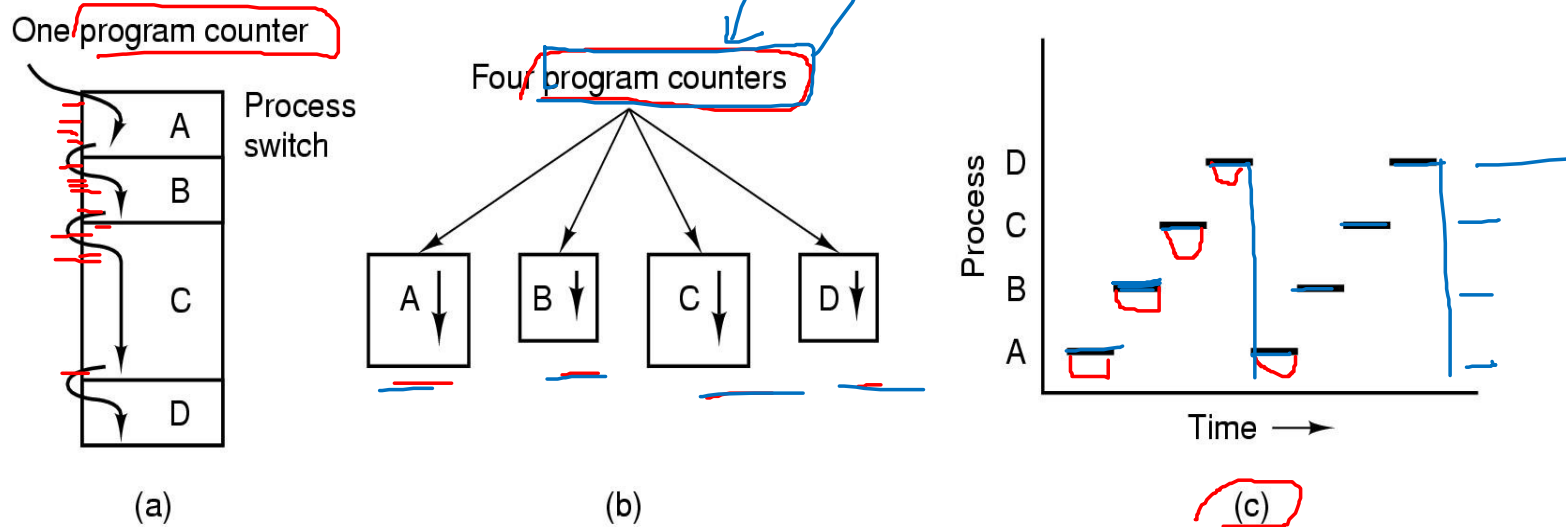
Contexto

- **Um S.O executa uma variedade de programas:**
 - Sistema Batch – JOBs
 - Sistema Time-shared – Prog.usuário ou tarefas
- **Processo:**
 - um programa em execução;
 - a execução dos processos deve progredir na forma seqüencial.
- **Um processo é constituído de:**
 - Contador de programa e conteúdo dos registradores;
 - Pilha (parâmetros, endereços de retorno....);
 - Seção de dados (variáveis globais);



Introdução a Processos

Introdução (Cont.)



- Multiprogramação de 4 programas;
- Mod. Conceitual de 4 processos seqüenciais independentes;
- Somente um processo ativo em algum instante

Introdução a Processos

Introdução (Cont.)

- Principais Eventos que causam a criação de um processo
 - Inicialização do sistema:
 - Execução de um sistema de criação de processos;
 - Usuário requisita a criação de um novo processo;
 - Inicialização de um Job Batch;

Introdução a Processos

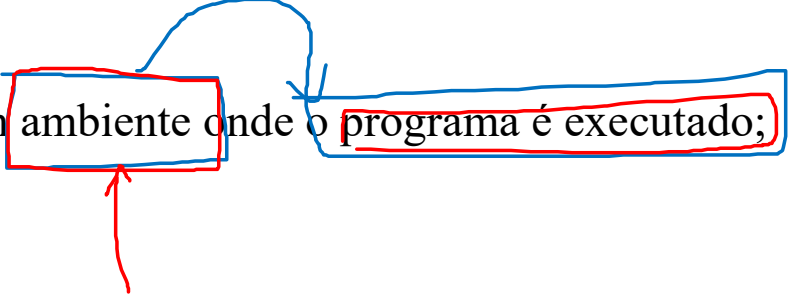
Introdução (Cont.)

- Condições que causam o termino de um processo:
 - Saída Normal (voluntário);
 - Saída por Erro (voluntário);
 - Erro Fatal (involuntário);
 - Finalizado por outro processo (involuntário).

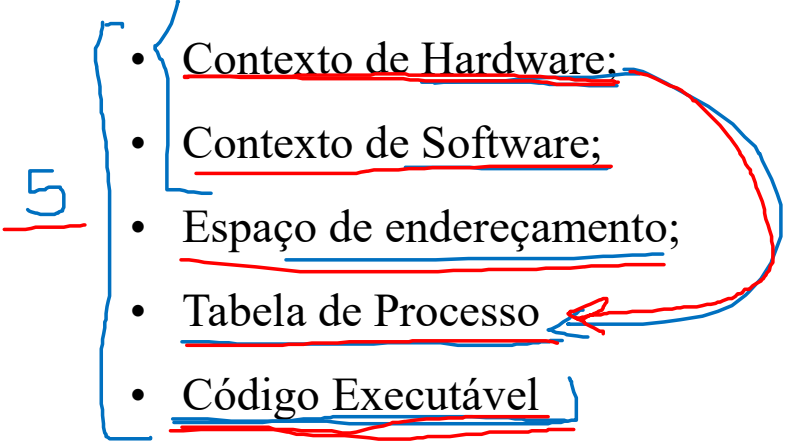
Introdução a Processos

Modelo do Processo

- **Conceito de Processo:**

- Pode ser definido como sendo um ambiente onde o programa é executado;
- 

- **Este ambiente é formado por:**

- 
- Contexto de Hardware;
 - Contexto de Software;
 - Espaço de endereçamento;
 - Tabela de Processo
 - Código Executável

Introdução a Processos

Modelo do Processo (Cont.)

- **Contexto de Hardware:**

- Conteúdo dos Registradores:

- Contador de programas (PC), Apontador de pilha (SP) e bits de estado.

- Fundamental para que os processos se revezem no processador;

- Implementação dos Sistemas de Tempo Compartilhado (time-sharing)

- Esta troca entre processos no processador é conhecido como:

- Mudança de Contexto (troca de um contexto por outro)

- Consiste em salvar o conteúdo dos registradores da CPU e

- Carregá-los com os valores referentes ao processo que esteja ganhando a utilização do CPU.

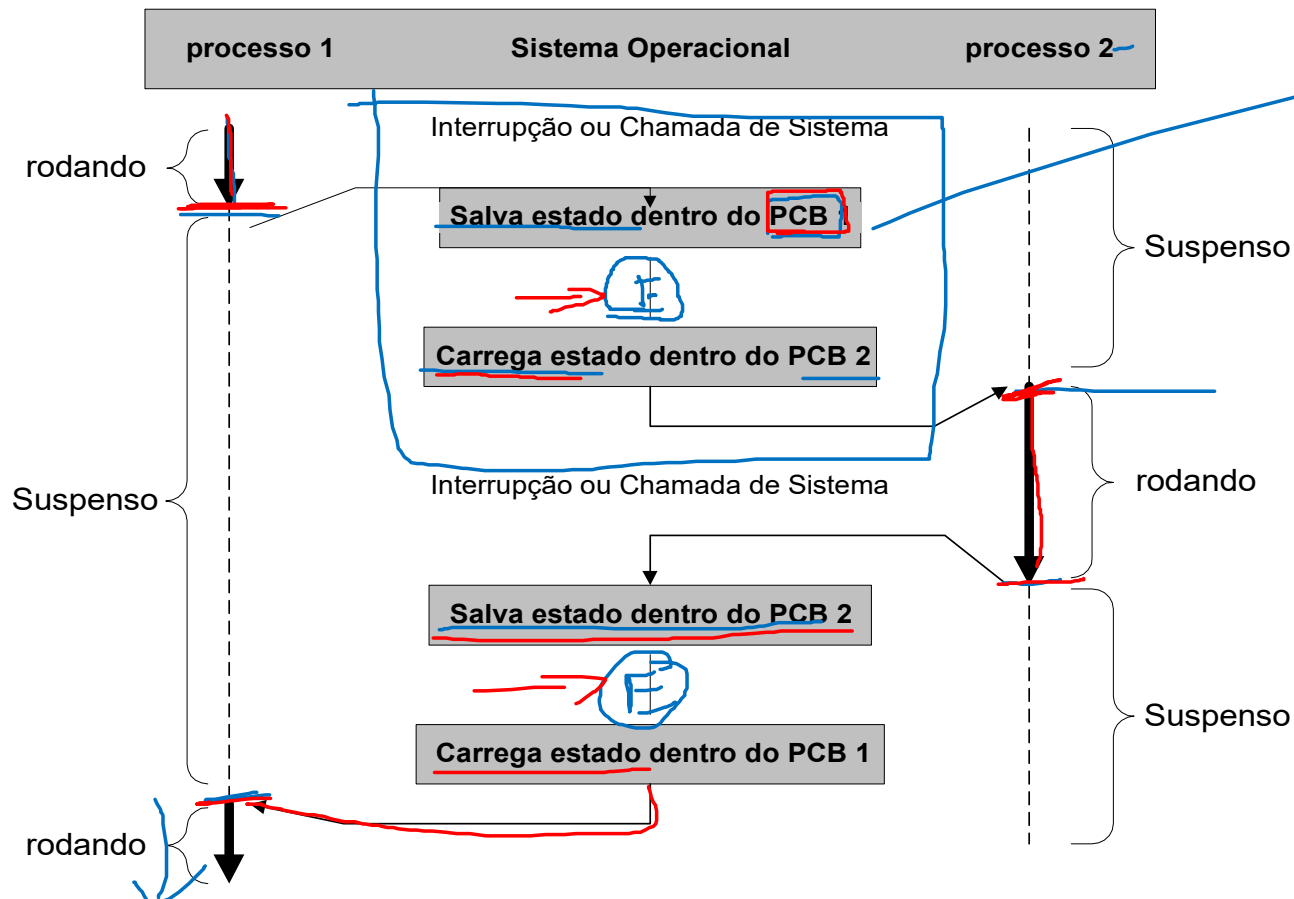
$\rightarrow P_A$

$\rightarrow P_B$

Introdução a Processos

Modelo do Processo (Cont.)

- Contexto de Hardware:



Introdução a Processos

Modelo do Processo (Cont.)

- **Contexto de Software:**

- Especifica características do processo que vão influir na execução de um programa;
 - São determinadas no momento da criação do processo, mas pode ser alterada;
- Exemplo: número máximo de arquivos abertos simultaneamente ou tamanho do buffer para operações de E/S.
- Define 3 grupos básicos de informações sobre um processo:
 - Identificação;
 - Quotas e privilégios;

Introdução a Processos

Modelo do Processo (Cont.)

- **Contexto de Software:**

- Identificação:

- Cada processo criado pelo sistema possui:
 - identificação única do processo (PID);
 - Identificação do usuário (UID) ou do processo que o criou;
 - Alguns sistemas, o nome também é utilizado;
 - O PID é utilizado pelo SO ou por outro usuários quando desejam referenciar a um determinado processo (p.ex. mudar características);
 - As identificações implementam um modelo de segurança;

Introdução a Processos

Modelo do Processo (Cont.)

- **Contexto de Software:**

- Quotas:

- Limite de cada recurso do sistema que um processo pode alocar;
 - Pode influenciar a execução do processo;
 - Exemplo:
 - Numero máximo de arquivos abertos simultaneamente;
 - Tamanho máximo de memória que o processador pode alocar;
 - Numero máximo de operações de E/S;
 - Numero máximo de processos e sub-processos que podem ser criados;

- Previlégios:

- Definem o que o processo pode ou não fazer em relação ao Sistema e outros processos.
 - Ex: eliminar processos, ter acesso a arquivos que não lhe pertence, operações e à gerencia do sistema etc..

Introdução a Processos

Modelo do Processo (Cont.)

- **PCB** - Bloco de Controle do Processo ou Tabela de Processo:
 - Cada processo é representado no S.O. por esta estrutura;
 - Mantém todas as informações referentes ao processo;

<u>Process management</u>	<u>Memory management</u>	<u>File management</u>
<u>Registers</u>	<u>Pointer to text segment</u>	<u>Root directory</u>
<u>Program counter</u>	<u>Pointer to data segment</u>	<u>Working directory</u>
<u>Program status word</u>	<u>Pointer to stack segment</u>	<u>File descriptors</u>
<u>Stack pointer</u>		<u>User ID</u>
<u>Process state</u>		<u>Group ID</u>
<u>Priority</u>		
<u>Scheduling parameters</u>		
<u>Process ID</u>		
<u>Parent process</u>		
<u>Process group</u>		
<u>Signals</u>		
<u>Time when process started</u>		
<u>CPU time used</u>		
<u>Children's CPU time</u>		
<u>Time of next alarm</u>		

Introdução a Processos

Modelo do Processo (Cont.)

- **Bloco de Controle do Processo (PCB):**
 - **Process State:**
 - O processo pode estar pronto, executando, esperando.
 - **Program Counter:**
 - Indica o próximo endereço da próxima instrução a ser executado neste processo;
 - **Registradores da CPU:**
 - Depende da arquitetura: acumuladores, índices de registros, ponteiro de pilha, registradores de propósito geral, dentre outros.
 - **Informações de escalonamento de CPU:**
 - Prioridade do processo, ponteiros para fila de escalonamento, dentre outros;
 - **Informações de gerenciamento de memória:**
 - Valores dos registradores Base e limite, tabela de páginas ou de segmentos, dentre outros;

Introdução a Processos

Estados do Processo

- **Sistemas Multiprogramáveis:**

- Durante seu ciclo de vida, o processo muda de estado várias vezes.

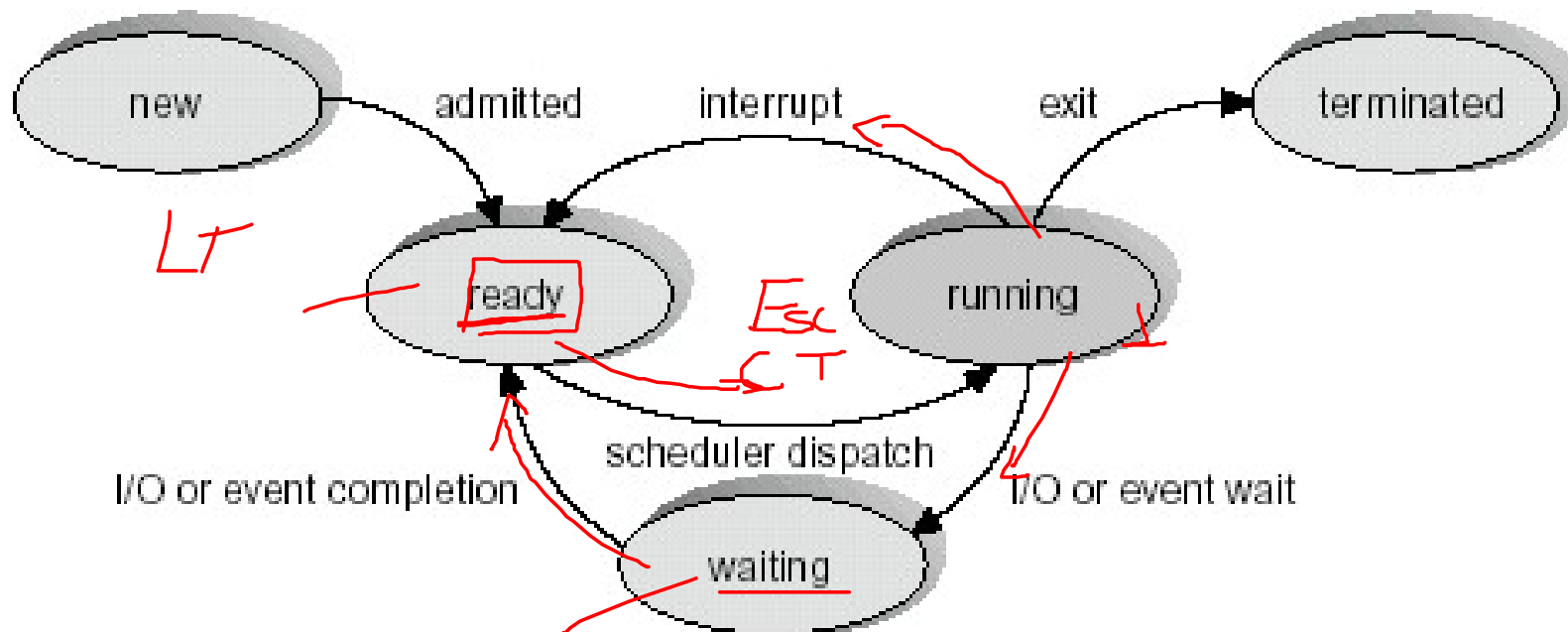
- Estados do processo:

- new: O processo é inicialmente criado.
 - running: Instruções são inicialmente executadas.
 - waiting: O processo está esperando algum evento ocorrer.
 - ready: O processo está esperando uma nova fatia de tempo no processador.
 - terminated: O processo finalizou sua execução.
- 5

Introdução a Processos

Estados do Processo (Cont.)

- Estados do processo:



Introdução a Processos

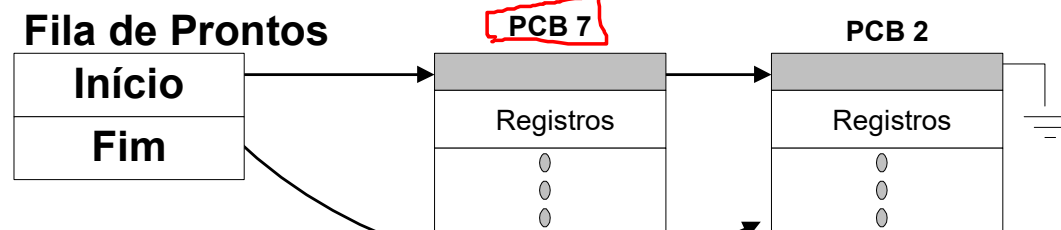
Estados do Processo (Cont.)

- Gerenciamento dos Processos pelo S.O.

- Através de Listas Encadeadas compostas por PCB's:

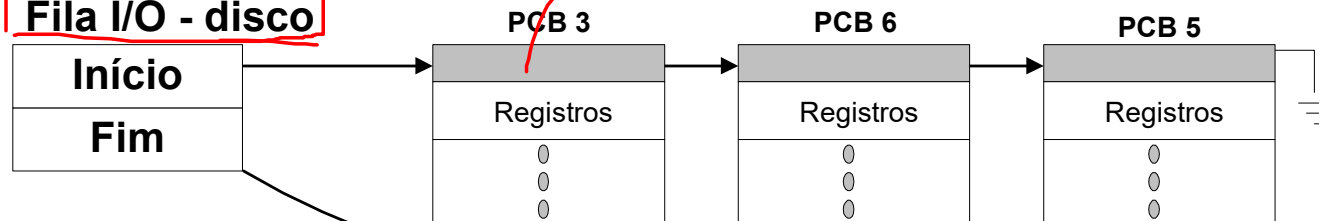
- Lista de Processos READY

Fila de Prontos



- Lista de Processos WAITING

Fila I/O - disco



Introdução a Processos

Estados do Processo (Cont.)

- **Tipos de Processos:**

- Podem ser classificados de acordo com o tipo de processamento

- CPU-BOUND

- Maior parte do tempo no estado de RUNING;
- Poucas Operações de E/S;
- Exemplo: aplicações matemáticas

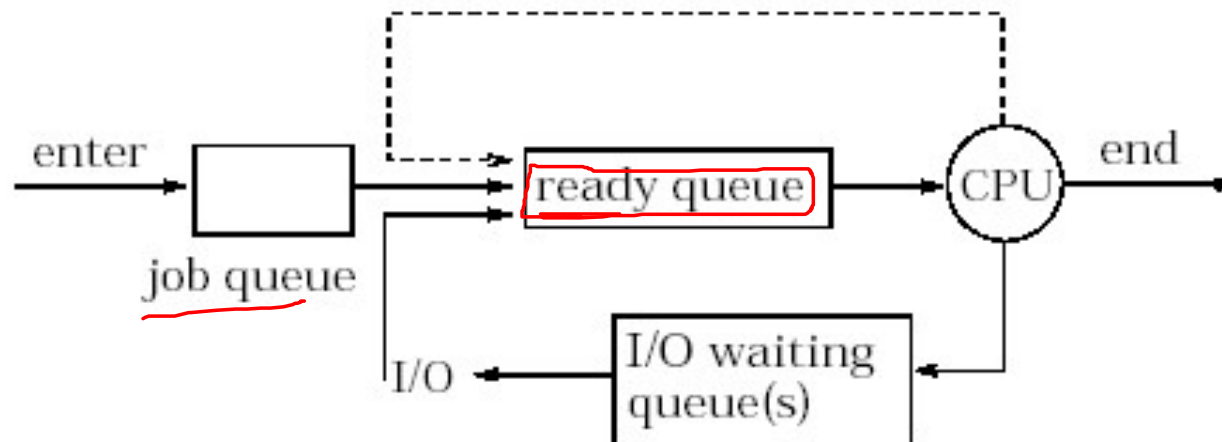
- I/O BOUND

- Maior parte do tempo no estado de WAITING;
- Elevado numero de operações de E/S;
- Exemplo: Aplicações comerciais;

Introdução a Processos

Escalonamento de Processos

- Job queue – Conjunto de todos os processos no sistema;
- Ready queue – Conjunto de todos os processos residentes na memória principal prontos e esperando para serem executados;
- Device queues – Conjunto de todos os processos esperando em um dispositivo de I/O (cada dispositivo tem uma fila própria);
- Processos migram entre as várias filas.



Introdução a Processos

Escalonamento de Processos (Cont.)

- **Escalonadores:**

- Responsável por escolher um processo para ser executado;

- Dividem-se em 3 tipos de escalonadores:

- Long-term scheduler (or job scheduler);
- Short-term scheduler (or CPU scheduler);
- Medium-term scheduler;

Introdução a Processos

Escalonamento de Processos (Cont.)

- **Escalonadores:**

- Long-term scheduler (or job scheduler)
 - Seleciona processos de um dispositivo de armazenamento (disco) e carrega-o para a memória para execução (fila de prontos).
 - É executado muita menos freqüente;
 - É responsável por controlar o grau de multiprogramação:
 - Numero de processos na memória;
 - Normalmente é executado quando um processo finaliza;

Introdução a Processos

Escalonamento de Processos (Cont.)

- **Escalonadores:**

Política

- **Short-term scheduler** (or CPU scheduler)

MC

- Seleciona, entre os processos da fila de prontos, qual será o próximo a ser executado e aloca a CPU para ele.
 - É invocado muito mais freqüentemente e muito mais rápido:
 - Pelo menos uma vez a cada 100 milisegundos;
 - Gasta 10 milisegundos para decidir qual processo executar
 - 9 % da CPU é gasta pelo trabalho de escalonamento;

Introdução a Processos

Escalonamento de Processos (Cont.)

- **Escalonadores:**

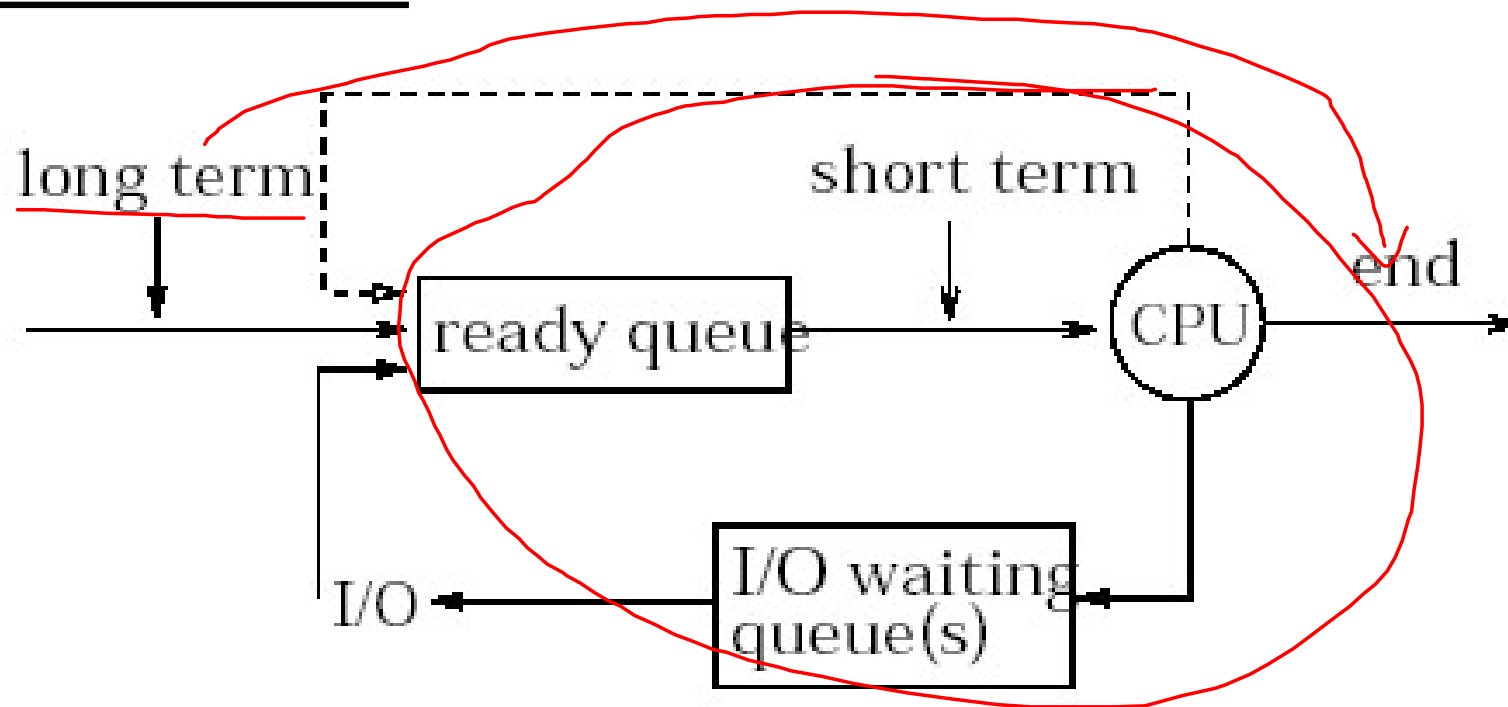
- **Medium-term scheduler:**

- Disponíveis em alguns sistemas de tempo compartilhado;
 - Normalmente utilizado para remover processos da memória e diminuir o grau da multiprogramação;
 - Principalmente utilizado durante **SWAPPING:**
 - O processo sai da memória e vai para o disco;
 - Mais tarde, o mesmo retorna para a memória e volta a executar de onde parou;
 - Swapping normalmente ocorre quando há falta de memória principal e um outro processo de maior prioridade deve ser executado;

Introdução a Processos

Escalonamento de Processos (Cont.)

- **Escalonadores:**



Introdução a Processos

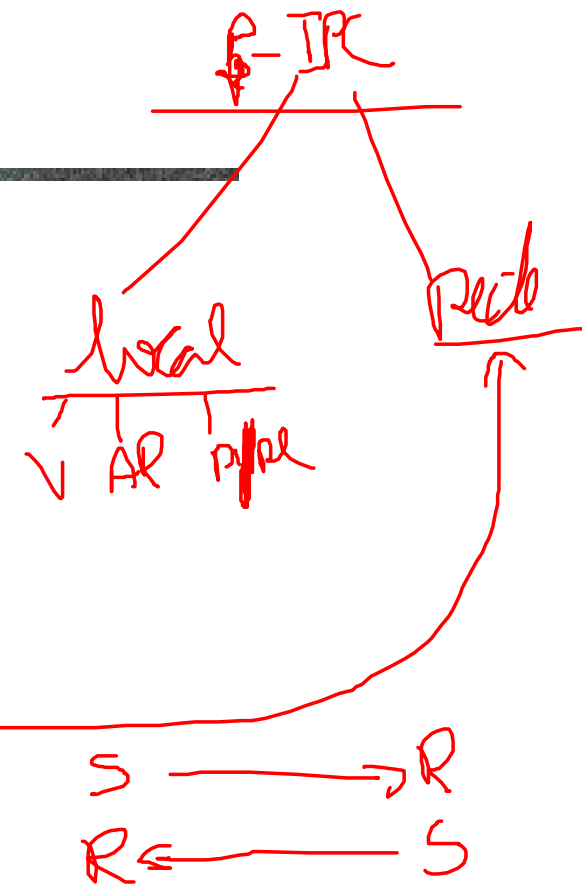
Comunicação Inter-Processos (IPC)

- IPC provê um mecanismo que permite que processos se comuniquem e que sincronizem suas ações;
- Utilizam sistema de troca de Mensagens:
 - Não há necessidade de variáveis compartilhadas;
- IPC é implementado por meio de duas operações:
 - send (mensagem) : tamanho da mensagem fixa ou variável
 - receive (mensagem)

Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Se P e Q esperam se comunicar, eles necessitam:
 - Estabelecer um link de comunicação entre eles;
 - Trocar mensagens via send/receive;
- Implementação de link de comunicação pode ser:
 - Físico (p.ex., shared memory, barramento via hardware)
 - Lógico (p.ex., propriedades lógicas - sockets)
- O link de comunicação entre processos pode ser:
 - Comunicação Direta;
 - Comunicação Indireta;



Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Comunicação Direta;
 - O processo deve saber o nome do outro processo explicitamente;
 - send (P,message) – Envia uma mensagem p/ o processo P;
 - receive (Q,message) – recebe uma mensagem do processo Q;
 - Propriedades dos Links de comunicação:
 - Os Links são estabelecidos automaticamente entre pares de processos (deve saber a identidade do outro).
 - Um link está associado exatamente entre um par de processos.
 - Entre cada par de processos existe exatamente um link.
 - O link deve ser unidirecional, mas pode ser bidirecional.

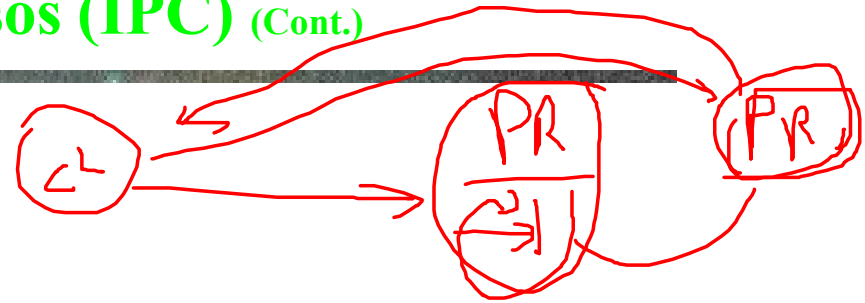
Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Comunicação Direta;

- Outra forma:

- send (P, mensagem) – Envia uma mensagem p/ o processo P;
- receive (ID mensagem) – recebe uma mensagem de algum processo;
- A variável ID será setada com o nome do processo com qual a comunicação foi estabelecida;



Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Comunicação Indireta:
 - As mensagens são enviadas e recebidas por meio de MAILBOX;
 - MAILBOX são visto abstratamente como um objeto onde processos colocam e removem mensagens;
 - Cada MAILBOX possui um identificador único (ID);
 - Processos se comunicam com outros processos apenas se compartilham um MAILBOX;

Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Comunicação Indireta:
 - Propriedades do link de Comunicação:
 - O Link é estabelecido somente se processo compartilha um MAILBOX comum;
 - Um link pode ser associado com mais de dois processos.
 - Entre cada par de processos se comunicando, pode haver vários vários Links de comunicação (um para cada MAILBOX).
 - Um Link pode ser unidirecional ou bidirecional.

Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Comunicação Indireta:
 - Operações:
 - Criar um novo MAILBOX
 - Enviar e Receber mensagens através do MAILBOX
 - Destruir um MAILBOX;

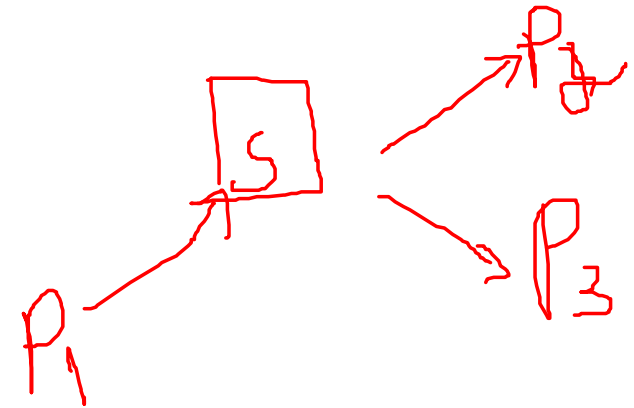
Introdução a Processos

Comunicação Inter-Processos (IPC) (Cont.)

- Comunicação Indireta: Exemplo:

- Mailbox compartilhado:

- P1, P2, e P3 compartilham mailbox A.
- P1 envia; P2 and P3 recebe.
- Quem recebe a mensagem de P1?



- Soluções

- Um link deve ser associado com mais de dois processos.
- Somente um processo por vez executa a operação recebe.
- O sistema seleciona arbitrariamente o receptor. Enviador será notificado quem recebeu.

**Chamadas de Sistema
(SVCs – SuperVisor Call)
para
Controle de Processos no Linux**

Introdução a Processos

Processos no Unix ou Linux

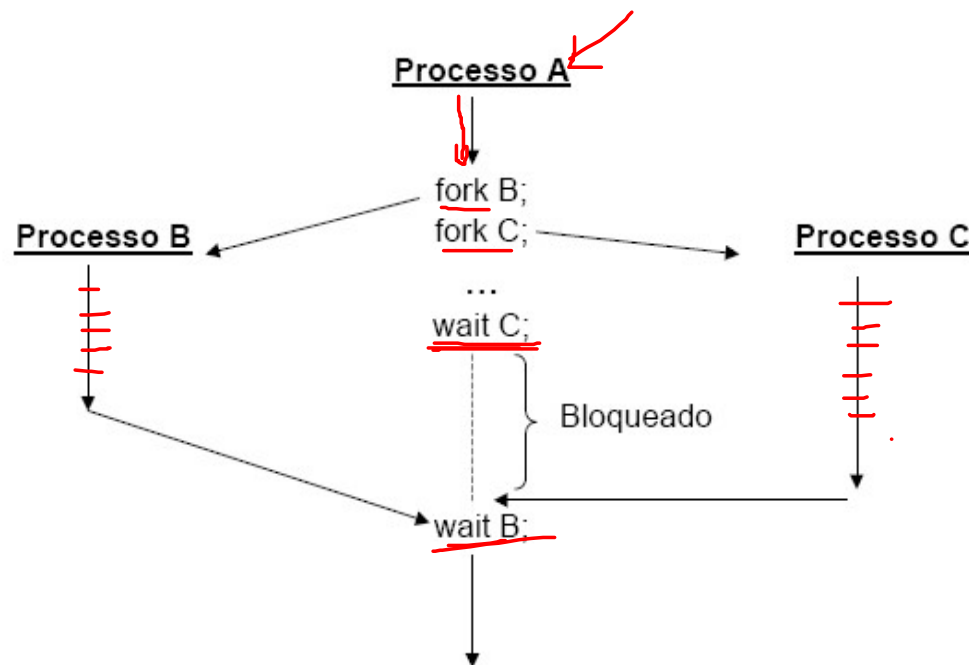
- **Processos no Linux:**
 - Criação e término
 - Hierarquia
 - Chamadas de sistema POSIX para gerenciamento de processos,
 - Implementação de aplicações multiprocessos

Introdução a Processos

Processos no Unix ou Linux

Criação de Processos

- A maioria dos sistemas operacionais usa um mecanismo de *spawn* para criar um novo processo a partir de um outro executável.



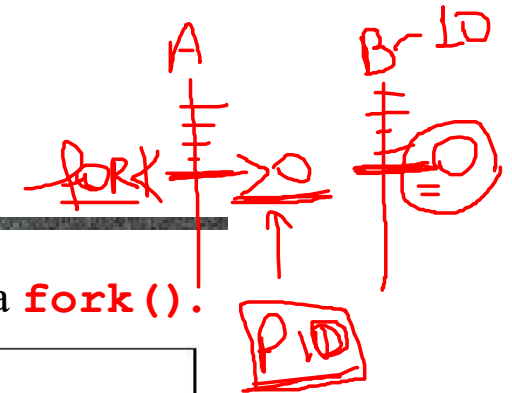
Introdução a Processos

Processos no Unix ou Linux

- No Linux, são usadas duas funções distintas relacionadas à criação e execução de programas. São elas:
 - `fork()` :
 - cria processo filho idêntico ao pai, exceto por alguns atributos e recursos.
 - `exec()` :
 - carrega e executa um novo programa.
- A sincronização entre processo pai e filho(s) é feita através da SVC `wait()`, que bloqueia o processo pai até que um processo filho termine.

Introdução a Processos

A SVC fork()



- A forma de se **criar um novo processo** é invocando a chamada **fork()**.

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

```
pid_t fork(void);
```

Retorna:

0 - para o processo filho
pid do filho - para o processo pai
-1 - se houve erro e o serviço não foi executado

- Fork() : **duplica/clona o processo** que executa a chamada.
 - O processo filho é uma cópia fiel do pai, ficando com uma cópia do segmento de dados, heap e stack (obs: o segmento de texto/código é muitas vezes partilhado por ambos).
- Processos **pai e filho continuam a sua execução** na instrução seguinte à chamada fork().
- Em geral, **não se sabe quem continua a executar imediatamente** após uma chamada a `fork()`, se é o pai ou o filho. Isso depende do algoritmo de escalonamento.

Introdução a Processos

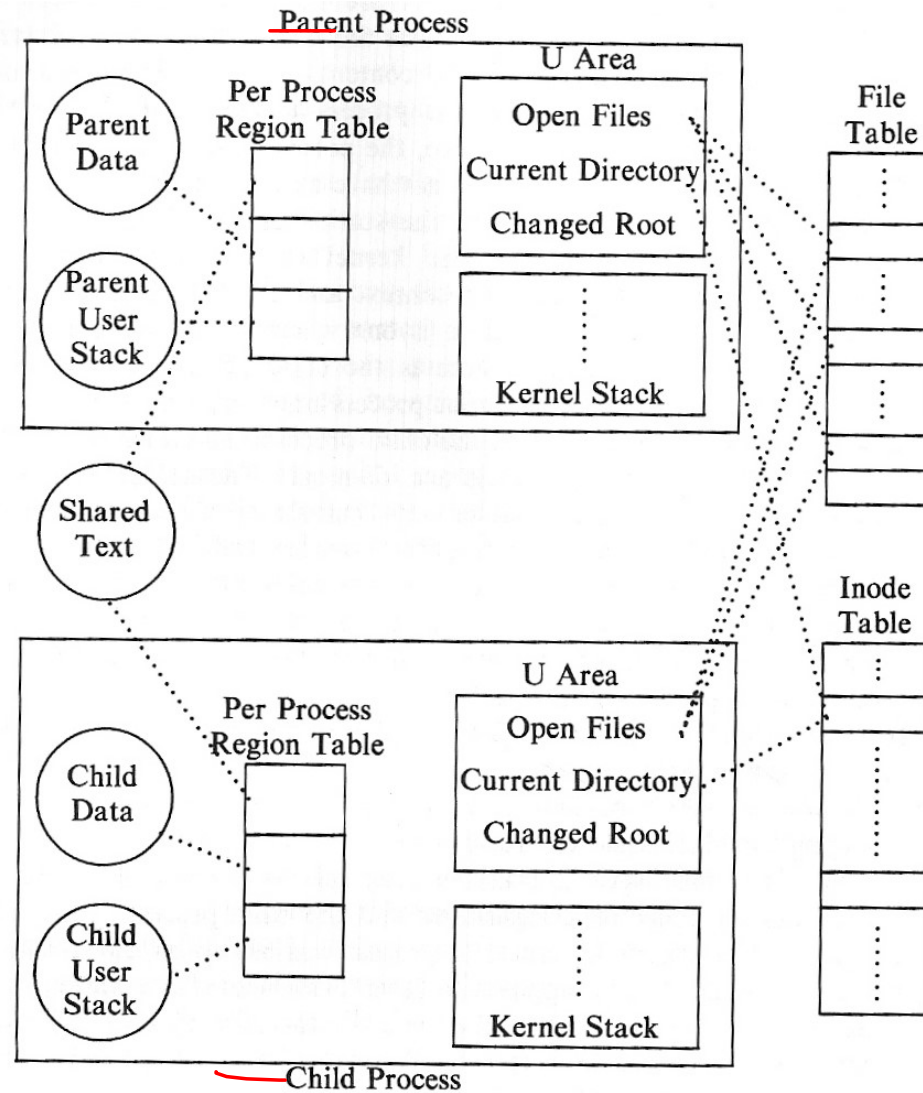
A SVC fork() (cont.)

■ O Processo Filho:

- Tem seu próprio espaço de endereçamento, com cópia de todas as variáveis do processo pai;
- herda do pai alguns atributos, tais como: variáveis de ambiente, variáveis locais e globais, privilégios e prioridade de escalonamento;
- herda alguns recursos, tais como arquivos abertos e devices.
- não são herdados pelo processo filho
 - atributos e recursos, tais como PID, PPID, sinais pendentes e estatísticas do processo.

A SVC *fork()*

(cont).



Introdução a Processos

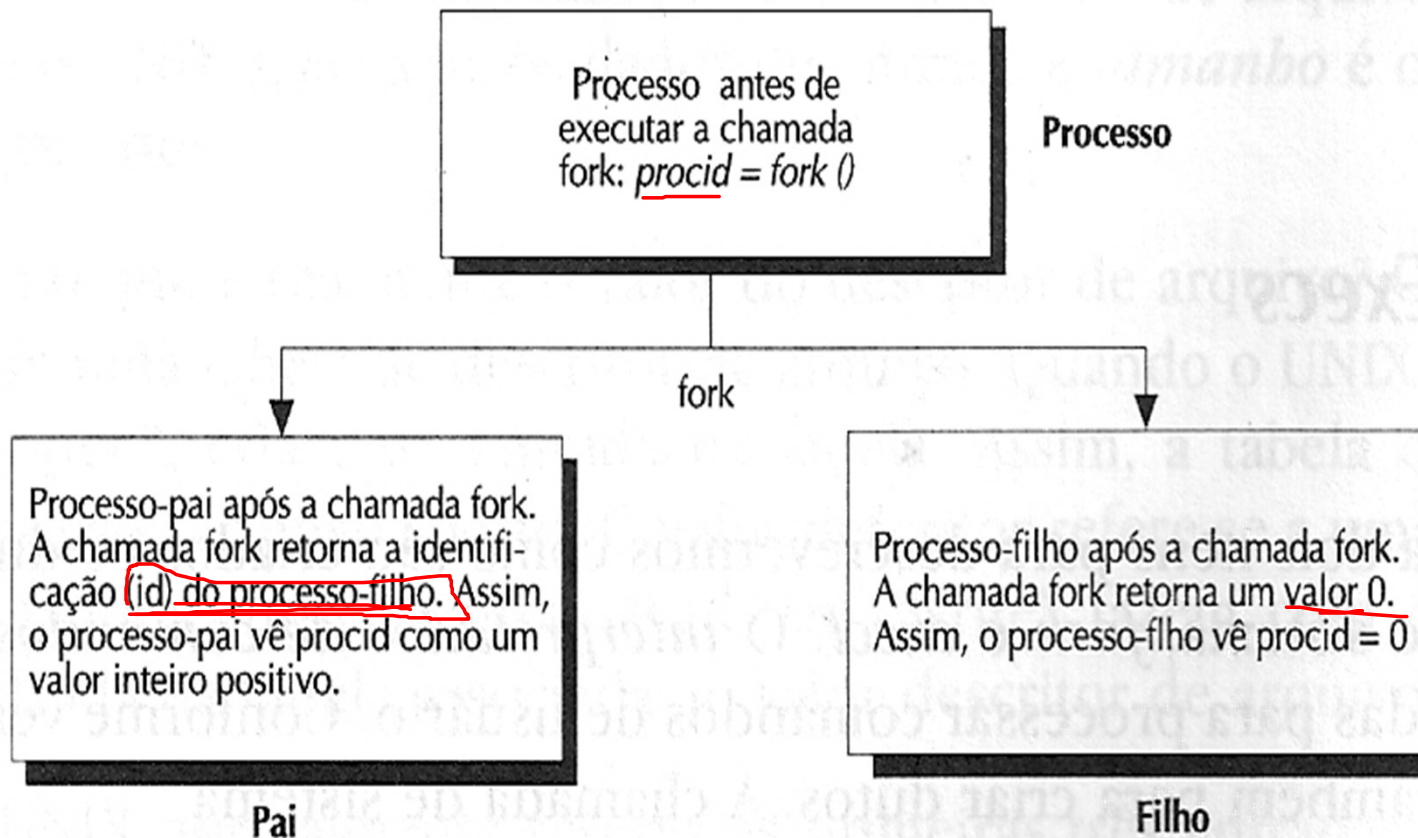
A SVC fork() (cont.)

X

- A função **fork()** é invocada uma vez (no processo-pai) mas retorna duas vezes:
 - uma no **Processo Pai** que a invocou
 - retorna o numero do *pid* do processo filho recém criado (*pid – process identifier*).
 - outra no novo **Processo Filho** criado.
 - retorna valor 0 (zero).

Introdução a Processos

A SVC fork() (cont.)



Introdução a Processos

A SVC fork() (cont.)

Estrutura Geral do fork()

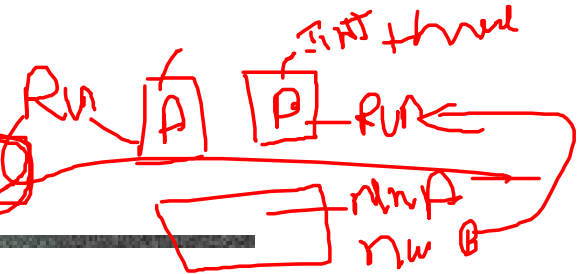
```
→ pid=fork() ;  
if (pid < 0) {  
    /* falha do fork */  
}  
else if (pid > 0) {  
    /* código do pai */  
}  
else { //pid == 0  
    /* código do filho */  
}
```

← A

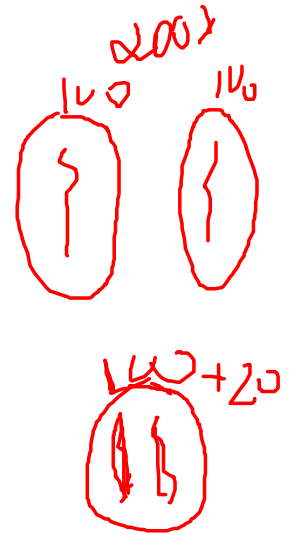
← B

Introdução a Processos

A SVC fork() (cont.) - Chamada Copy-on-Write



- Como **alternativa a** **significante ineficiência** do `fork()`, no Linux o `fork()` é implementado usando uma **técnica chamada copy-on-write (COW)**.
- Essa técnica **atrasa ou evita a cópia dos dados**.
 - Ao invés de copiar o espaço de endereçamento do processo pai, ambos podem **compartilhar uma única cópia somente de leitura**.
 - Se uma **escrita é feita**, **uma duplicação** é realizada e cada processo recebe uma cópia.
 - Consequentemente, a duplicação é feita apenas quando necessário, economizando tempo e espaço.
- O único ***overhead*** inicial do `fork()` é a **duplicação da tabela de páginas** do processo pai e a criação de um novo *proc Struct* (c/ PID para o filho).



Introdução a Processos

Identificação do Processo no UNIX

- Como visto, todos os **processos em Linux têm um identificador**, geralmente designados por **pid** (*process identifier*).
 - Os identificadores são **números inteiros** diferentes para cada processo (ou melhor, do tipo `pid_t` definido em **sys/types.h**).
- É sempre possível a um processo conhecer o seu próprio identificador e o do seu pai. Os **serviços a utilizar** para conhecer pid's (além do serviço `fork()`) são: **getpid(void)** e **getppid(void)**.

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

pid_t getpid(void);          /* obtém o seu próprio pid */
pid_t getppid(void);        /* obtém o pid do pai */
```

Estas funções são sempre bem sucedidas.

Introdução a Processos

User ID e Group ID

- Cada processo tem de um proprietário:
- UID (*User Identifier*) e GID (*Group Identifier*). Os nomes dos usuários e dos grupos servem apenas para facilitar o uso humano do computador.
- Cada usuário precisa pertencer a um ou mais grupos. Como cada processo (e cada arquivo) pertence a um usuário, logo esse processo pertence ao grupo de seu proprietário. Assim sendo, cada processo está associado a um UID e a um GID.
- Os números UID e GID variam de 0 a 65536. Dependendo do sistema, o valor limite pode ser maior. No caso do usuário root, esses valores são sempre 0 (zero). Assim, para fazer com que um usuário tenha os mesmos privilégios que o root, é necessário que seu GID seja 0.

Introdução a Processos

User ID e Group ID (cont.)

- Primitivas:

- P/ user: `uid_t getuid(void)` / `uid_t geteuid(void)`
- P/ group: `gid_t getgid(void)` / `gid_t getegid(void)`

- Comandos úteis:

- `id`: lista os ID's do usuário e do seu grupo primário. Lista também todos os outros grupos nos quais o usuário participa.

- Arquivos úteis:

- /etc/passwd
- /etc/group

- Formato do arquivo /etc/passwd:

- usuário:senha:UID:GID:grupo primário do usuário:nome do usuário:diretório home:shell inicial

- Formato do arquivo /etc/group:

- grupo:senha:GID:lista dos usuários do grupo

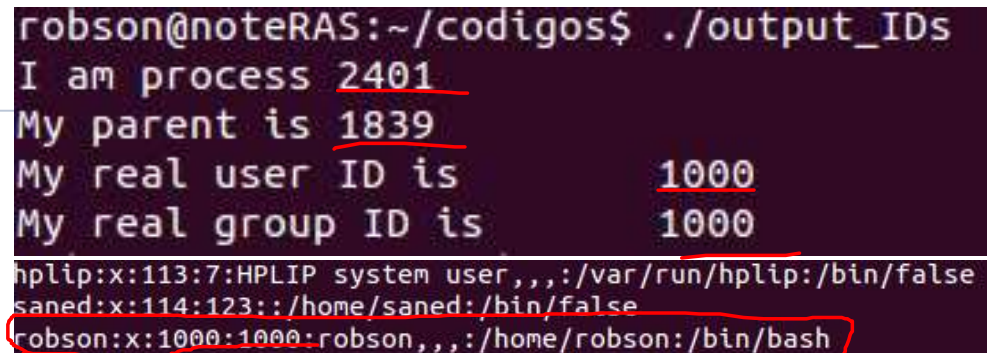
Exemplo 1 - Exibindo PID's (arquivo output_IDs.c)

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

int main (void) {
    printf("I am process %ld\n", (long) getpid());
    printf("My parent is %ld\n", (long) getppid());

    printf("My real user ID is          %5ld\n", (long) getuid());
    printf("My real group ID is         %5ld\n", (long) getgid());

    return 0;
}
```



```
robson@noteRAS:~/codigos$ ./output_IDs
I am process 2401
My parent is 1839
My real user ID is          1000
My real group ID is         1000
hplip:x:113:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:114:123:./home/saned:/bin/false
robson:x:1000:1000:robson,,,:/home/robson:/bin/bash
```

Exemplo 2: Fork Simples (arquivo simple_fork.c)

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

```
int main(void) {
    int x;
```

```
    x = 0;
```

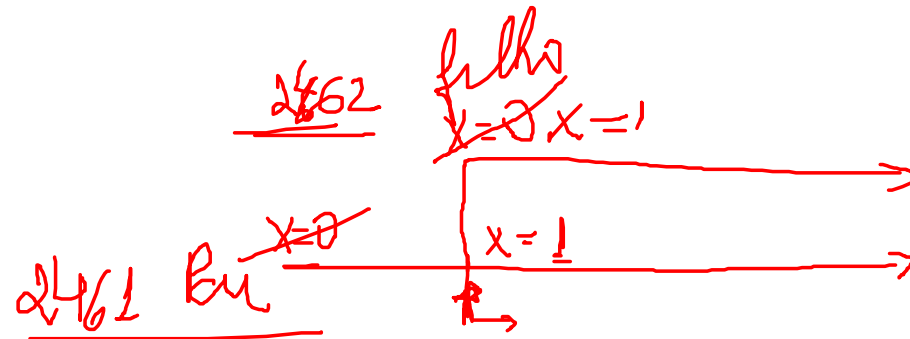
```
    fork();
```

```
    x = 1;
```

```
    printf("I am process %ld and my x is %d\n",
           (long) getpid(), x);
```

```
    return 0;
```

```
}
```



```
robson@noteRAS:~/codigos$ gcc simple_fork.c -o simple_fork
robson@noteRAS:~/codigos$ ./simple_fork
```

```
I am process 2461 and my x is 1
robson@noteRAS:~/codigos$
I am process 2462 and my x is 1
```



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

```
int glob = 6;
int main(void) {
    int var;          /* external variable in initialized data */
    pid_t pid;        /* automatic variable on the stack */
    var = 88;
    printf("before fork\n");
    if ( (pid = fork()) < 0)
        fprintf(stderr, "fork error\n");
    else if (pid == 0) { /* ***child*** */
        glob++;         /* modify variables */
        var++;
    }
    else
        sleep(30);      /* ***parent***; try to guarantee that child ends first */
    printf("getpid = %d, getppid = %d, glob = %d, var = %d, pid = %d\n",
        getpid(), getppid(), glob, var, pid);
    return 0;
}
```

Ex.3 - Diferenciando Pai e Filho (arquivo two_procs.c)

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

```
robson@noteRAS:~/codigos$ ./two_procs
before fork
getpid = 2534, getppid = 2533, glob = 7, var = 89, pid = 0
getpid = 2533, getppid = 1839, glob = 6, var = 88, pid = 2534
```

Exemplo 4 - mypid x gettpid (arquivo myPID.c)

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

```
int main(void) {
    pid_t childpid;
    pid_t mypid;

    mypid = getpid();
    childpid = fork();
    if (childpid == -1) {
        perror("Failed to fork");
        return 1;
    }
    if (childpid == 0)
        printf("I am child %ld, ID = %ld\n", (long int) getpid(), (long int) mypid);
    else
        printf("I am parent %ld, ID = %ld\n", (long int) getpid(), (long int) mypid);
    return 0;
}
```

```
robson@noteRAS:~/codigos$ gcc myPID.c -o myPID
robson@noteRAS:~/codigos$ ./myPID
Eu sou o Pai 2378, ID do Pai = 2378
robson@noteRAS:~/codigos$
Eu sou o Filho 2379, ID do Pai = 2378
```

Exemplo 5 – Simple Chain

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

```
int main ((int argc, char *argv[]) {
    pid_t childpid = 0;
    int i, n;
```

```
    if (argc != 2) { /* check for valid number of command-line arguments */
        fprintf(stderr, "Usage: %s processes\n", argv[0]);
        return 1;
    }
```

```
    n = atoi(argv[1]);
```

```
    for (i = 1; i < n; i++)
```

```
        if (childpid = fork()) //only the father (or error) enters
            break;
```

```
    fprintf(stderr, "i:%d process ID:%ld parent ID:%ld child ID:%ld\n",
               i, (long)getpid(), (long)getppid(), (long)childpid);
    return 0;
```

```
}
```

```
robson@noteRAS:~/codigos$ ./simple_chain 5
i:1 process ID:2609 parent ID:1839 child ID:2610
robson@noteRAS:~/codigos$ i:2 process ID:2610 parent ID:1 child ID:2611
i:3 process ID:2611 parent ID:1 child ID:2612
i:4 process ID:2612 parent ID:1 child ID:2613
i:5 process ID:2613 parent ID:1 child ID:0
```

Exemplo 6 – Simple Fan (arquivo simple_fan.c)

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

```
int main (int argc, char *argv[]) {
    pid_t childpid = 0;
    int i, n;
```

```
    if (argc != 2){ /* check for valid number of command-line arguments */
        fprintf(stderr, "Usage: %s processes\n", argv[0]);
        return 1;
    }
```

```
    n = atoi(argv[1]);
```

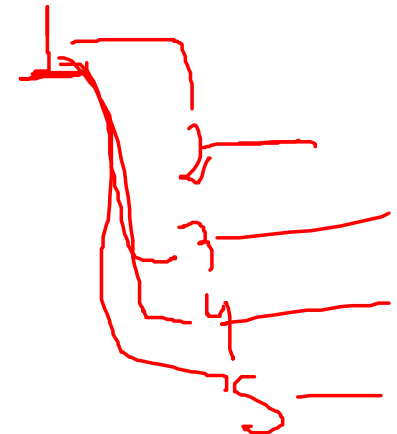
```
    for (i = 1; i < n; i++)
```

```
        if ((childpid = fork()) <= 0) //only the child (or error) enters
            break;
```

```
    fprintf(stderr, "i:%d process ID:%ld parent ID:%ld child ID:%ld\n",
               i, (long)getpid(), (long)getppid(), (long)childpid);
    return 0;
}
```

P → D 1

```
robson@noteRAS:~/codigos$ ./simple_fan 5
i:5 process ID:2649 parent ID:1839 child ID:2653
robson@noteRAS:~/codigos$ i:4 process ID:2653 parent ID:1 child ID:0
i:3 process ID:2652 parent ID:1 child ID:0
i:2 process ID:2651 parent ID:1 child ID:0
i:1 process ID:2650 parent ID:1 child ID:0
```

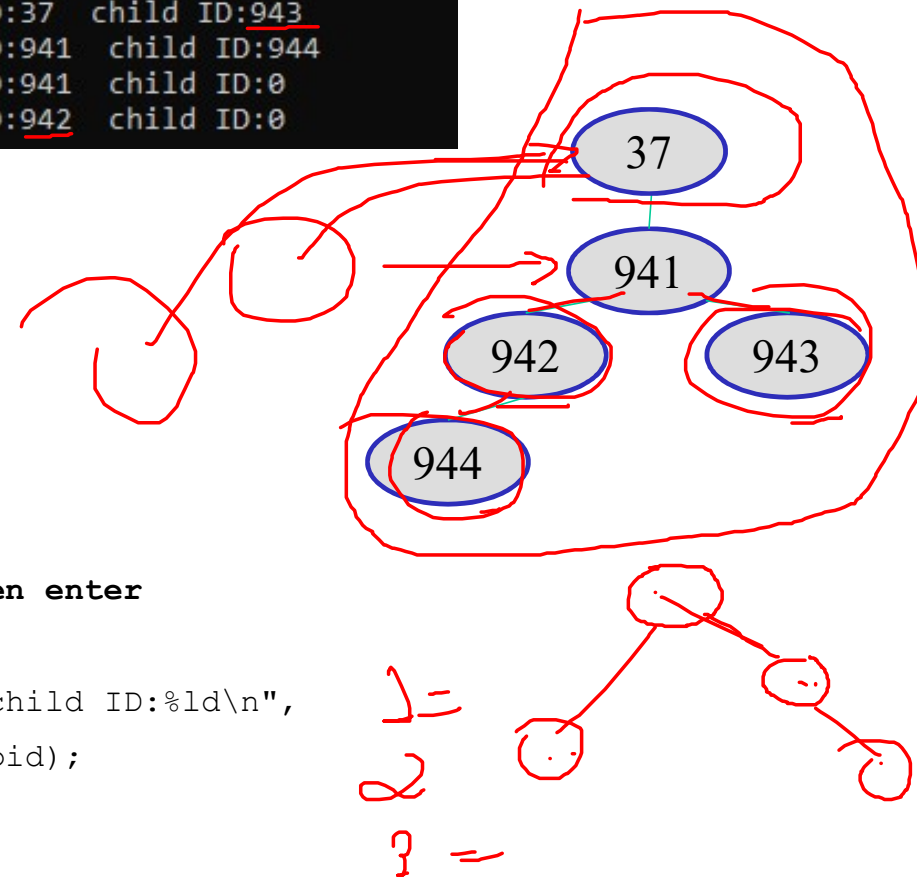


Exemplo 7 – Chain Geral (arquivo chain_geral.c)

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

```
int main (int argc, char *argv[]) {
    pid_t childpid = 0;
    int i, n;
    /* check for valid number of command-line arguments */
    n = atoi(argv[1]);
    for (i = 1; i < n; i++)
        if ((childpid = fork()) == -1) // father and children enter
            break;
    fprintf(stderr, "i:%d process ID:%ld parent ID:%ld child ID:%ld\n",
            i, (long)getpid(), (long)getppid(), (long)childpid);
    return 0;
}
```

```
robson@RAS-NOTE:~/Slides$ ./chain_geral 3
parent ID:37
i:3 process ID:941 parent ID:37 child ID:943
i:3 process ID:942 parent ID:941 child ID:944
i:3 process ID:943 parent ID:941 child ID:0
i:3 process ID:944 parent ID:942 child ID:0
```



O Comando ps

(retirado de *man ps*) *By default, ps selects all processes with the same effective user ID (euid=EUID) as the current user and associated with the same terminal as the invoker. It displays process ID (pid=PID), terminal associated with the process (tname=TTY), cumulated CPU time in [dd-]hh:mm:ss format (time=TIME), and the executable name (ucmd=CMD). Output is unsorted by default.*

Alguns tributos:

- a Lista todos os processos
- e Mostra as variáveis associadas aos processos
- f Mostra a árvore de execução dos processos
- l Mostra mais campos
- u Mostra o nome do usuário e a hora de início
- x Mostra os processos que não estão associados a terminais
- t Mostra todos os processos do terminal

Opções interessantes:

- \$ ps Lista os processos do usuário associados ao terminal
- \$ ps l Idem, com informações mais completas
- \$ ps a Lista também os processos não associados ao terminal
- \$ ps u Lista processos do usuário
- \$ ps U <user> ou \$ps -u <user> Lista processos do usuário <user>
- \$ ps p <PID> Lista dados do processo PID
- \$ ps r Lista apenas os processos no estado running
- \$ ps al, \$ ps ux, \$ ps au, \$ ps aux

O Comando ps (cont.)

robson@noteRAS:~/codigos\$ ps

PID	TTY	TIME	CMD
2464	pts/0	00:00:00	bash
2885	pts/0	00:00:00	ps

robson@noteRAS:~/codigos\$ ps -la

	F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
0	<u>S</u>		1000	2608	2592	0	80	0	-	888	wait	pts/1	00:00:00	man
0	<u>S</u>		1000	2618	2608	0	80	0	-	847	n_tty_	pts/1	00:00:00	pager
0	<u>R</u>		1000	2878	2464	0	80	0	-	626	-	pts/0	00:00:00	ps

- F-flags, S-state, C-CPU utilizado para escalonamento (uso muito baixo é reportado como zero),
- NI-nice value, ADDR- endereço memoria do processo;
- SZ-tamanho em blocos do processo
- WCHAN-rotina do kernel onde o processo dorme (processos em execução são marcados com hífen),
- TIME- cumulative execution time, CMD-command name.

```

samuel@houston:~$ ps ux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
spereira  6364  0.2  0.6 7892 1552 ?        S      10:12   0:00 sshd: spereira@pts/3
spereira  6365  0.5  1.2 5588 3080 pts/3    Ss     10:12   0:00 -bash
spereira  6398  0.2  0.6 7892 1552 ?        S      10:13   0:00 sshd: spereira@pts/4
spereira  6399  1.9  1.1 5584 3068 pts/4    Ss     10:13   0:00 -bash
spereira  6418  0.5  0.7 4432 1828 pts/4    S+     10:13   0:00 vi
spereira  6423  0.0  0.3 2560  996 pts/3    R+     10:13   0:00 ps -U spereira u

```

O Comando ps (cont.)

- **USER** – Nome do Utilizador
- **PID** – Número do Processo
- **%CPU** – Percentagem de utilização do CPU
- **%MEM** – Percentagem de utilização de memória
- **VSZ** – Tamanho da memória virtual
- **RSS** – Tamanho da memória residente
- **TTY** – Número do terminal, se tivermos várias sessões, temos diferentes terminais
- **STAT** – Estado actual do processo
- **START** – Hora em que o processo foi iniciado
- **TIME** – Tempo de processamento consumido
- **COMMAND** – Nome do comando

Estados dos Processos

D	Processo em "Uninterruptible sleep"
R	Processo a correr
S	Processo suspenso
T	Processo parado
X	Processo morto (não aparece na lista dos processos)
Z	Processo <i>Zombie</i> , está terminado, mas está ligado pelo processo que o iniciou

<	Corre em alta prioridade
N	Corre em baixa prioridade
L	Aloca as páginas na memória
s	Líder de sessão, garante que o processo termina quando o user faz <i>logout</i>
l	Processo em multi-thread
+	Corre em <i>foreground</i>

Introdução a Processos

Processos **Zombie**

- Um processo só pode terminar se o seu pai aceitar o seu **código de terminação**:
 - valor retornado por `main()` ou passado a `exit()`, através da execução de uma chamada aos serviços `wait()` / `waitpid()`.
- Um processo que terminou, mas cujo pai ainda não executou um dos `wait`'s passa ao **estado "zombie"**.
 - Na saída do comando `ps` o estado destes processos aparece como `Z` e o seu nome identificado como `<defunct>`.
- Processo no estado de zombie:
 - **memória é liberada** mas permanece no sistema alguma informação sobre ele (processo continua **ocupando a tabela de processos** do kernel).
- Se o processo pai terminar antes do filho, esse torna-se **órfão** e é adotado pelo processo `init` (`PID=1`).

Exemplo 1 – Zombie(1) (arquivo testa_zombie_1.c)

```
/* rodar o programa em background */
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
    int pid ;
    printf("Eu sou o processo pai, PID = %d, e eu vou criar um filho.\n",getpid()) ;
    printf("Agora estou entrando em um loop infinito. Tchau!\n") ;
    pid = fork() ;
    if(pid == -1) /* erro */
    {
        perror("E impossível criar um filho") ;
        exit(-1) ;
    }
    else if(pid == 0) /* filho */
    {
        printf("Eu sou o filho, PID = %d. Estou vivo mas vou dormir um pouco. Enquanto isso, use o comando ps -l para conferir o meu PID, o meu estado (S=sleep), o PID do meu pai e o estado do meu pai (R=running). Daqui a pouco eu acordo.\n",getpid()) ;
        sleep(60) ;
        printf("Acordei! Vou terminar agora. Confira novamente essas informações. Nãããoooooooo!!! Virei um zumbi!!!\n") ;
        exit(0) ;
    }
    else /* pai */
    {
        for(;;) ; /* pai bloqueado em loop infinito */
    }
}
```

Exemplo 1 – Zombie(1) (arquivo testa_zombie_1.c)

`./testa_zombie_1 &`

```
robson@noteRAS:~/codigos$ Eu sou o processo pai, PID = 2910, e eu vou criar um filho.

Agora estou entrando em um loop infinito. Tchau!
Eu sou o filho, PID = 2911. Estou vivo mas vou dormir um pouco. Enquanto isso, use o comando ps -l para
conferir o meu PID, o meu estado (S=sleep), o PID do meu pai e o estado do meu pai (R=running). Daqui a
pouco eu acordo.

robson@noteRAS:~/codigos$ ps -l
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  1839  1829  0  80   0  -  1933 wait  pts/2      00:00:00 bash
0 R  1000  2910  1839  97  80   0  -   502 -    pts/2      00:00:06 testa_zombie_1
1 S  1000  2911  2910  0  80   0  -   502 hrtim pts/2      00:00:00 testa_zombie_1
0 R  1000  2912  1839  0  80   0  -   1177 -    pts/2      00:00:00 ps
robson@noteRAS:~/codigos$ Acordei! Vou terminar agora. Confira novamente essas informaões. Noooooooooo!!
Virei um zumbi!!!

robson@noteRAS:~/codigos$ ps -l
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  1839  1829  0  80   0  -  1933 wait  pts/2      00:00:00 bash
0 R  1000  2910  1839  99  80   0  -   502 -    pts/2      00:00:13 testa_zombie_1
1 Z  1000  2911  2910  0  80   0  -     0 exit  pts/2      00:00:00 testa_zombie_1 <defunct>
0 R  1000  2913  1839  0  80   0  -   1177 -    pts/2      00:00:00 ps
```

Exemplo 2 – Zombie(2) (arquivo testa_zombie_2.c)

```
/* rodar o programa em foreground */
```

```
#include <errno.h>
```

```
#include <signal.h>
```

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

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

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
    int pid ;
```

```
    printf("Eu sou o processo pai, PID = %d, e eu vou criar um filho.\n",getpid()) ;
```

```
    printf("Bem, eu já coloquei mais um filho no mundo e agora vou terminar. Fui!\n") ;
```

```
    pid = fork() ;
```

```
    if(pid == -1) /* erro */
```

```
    {
```

```
        perror("E impossivel criar um filho") ;
```

```
        exit(-1) ;
```

```
    }
```

```
    else if(pid == 0) /* filho */
```

```
    {
```

```
        printf("Eu sou o filho, PID = %d. Estou vivo mas vou dormir um pouco. Use o comando ps -l para conferir o meu estado (S=sleep) e o PID do meu pai. Notou algo diferente no PID do meu pai? Notou que eu não virei um zumbi? Daqui a pouco eu acordo.\n",getpid()) ;
```

```
        sleep(60) ;
```

```
        printf("Acordei! Vou terminar agora. Use ps -l novamente.\n") ;
```

```
        exit(0) ;
```

```
    }
```

```
    else /* pai */
```

```
    {
```

```
        /*   pai bloqueado em loop infinito */
```

```
    }
```

```
}
```

Exemplo 2 – Zombie(2) (arquivo testa_zombie_2.c)

`./testa_zombie_2 &`

```
robson@noteRAS:~/codigos$ ./testa_zombie_2&
[2] 2936
robson@noteRAS:~/codigos$ Eu sou o processo pai, PID = 2936, e eu vou criar um filho.
Bem, eu já coloquei mais um filho no mundo e agora vou terminar. Fui!
Eu sou o filho, PID = 2937. Estou vivo mas vou dormir um pouco. Use o comando ps -l para conferir o meu
estado (S=sleep) e o PID do meu pai. Notou algo diferente no PID do meu pai? Notou que eu não virei um z
umbi? Daqui a pouco eu acordo.

[2]+  Fim da execução com status 121      ./testa_zombie_2
robson@noteRAS:~/codigos$ ps -l
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000   1839   1829  0  80   0  -  1933 wait  pts/2        00:00:00 bash
0 R   1000   2910   1839  98  80   0  -   502 -    pts/2        00:03:14 testa_zombie_1
1 Z   1000   2911   2910  0  80   0  -     0 exit  pts/2        00:00:00 testa_zombie_1 <defunct>
1 S   1000   2937     1  0  80   0  -   502 hrtime pts/2        00:00:00 testa_zombie_2
0 R   1000   2938   1839  0  80   0  -   1177 -    pts/2        00:00:00 ps
robson@noteRAS:~/codigos$ Acordei! Vou terminar agora. Use ps -l novamente.
robson@noteRAS:~/codigos$ ps -l
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000   1839   1829  0  80   0  -  1933 wait  pts/2        00:00:00 bash
0 R   1000   2910   1839  98  80   0  -   502 -    pts/2        00:03:25 testa_zombie_1
1 Z   1000   2911   2910  0  80   0  -     0 exit  pts/2        00:00:00 testa_zombie_1 <defunct>
0 R   1000   2939   1839  0  80   0  -   1177 -    pts/2        00:00:00 ps
```

Exemplo 3 – Zombie(3) (arquivo testa_zombie_3.c)

```
/* rodar em background */
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{
    int pid ;
    printf("Eu sou o processo pai, PID = %d, e eu vou criar um filho.\n",getpid()) ;
    printf("Bem, agora eu vou esperar pelo término da execução do meu filho. Tchau!\n") ;
    pid = fork() ;
    if(pid == -1) /* erro */
    {
        perror("E impossivel criar um filho") ;    exit(-1) ;
    }
    else if(pid == 0) /* filho */
    {
        printf("Eu sou o filho, PID = %d. Estou vivo mas vou dormir um pouco. Use o comando ps -l para conferir o meu estado e o do meu pai. Daqui a pouco eu acordo.\n",getpid()) ;
        sleep(60) ;
        printf("Sou eu de novo, o filho. Acordei mas vou terminar agora. Use ps -l novamente.\n") ;
        exit(0) ;
    }
    else /* pai */
    {
        wait(NULL) ; /* pai esperando pelo término do filho */
    }
}
```


Exemplo 3 – Zombie(3) (arquivo testa_zombie_3.c)

Use o & no final do comando

```
robson@noteRAS:~/codigos$ ./testa_zombie_3&
[2] 2941
robson@noteRAS:~/codigos$ Eu sou o processo pai, PID = 2941, e eu vou criar um filho.
Bem, agora eu vou esperar pelo termino da execucao do meu filho. Tchau!
Eu sou o filho, PID = 2942. Estou vivo mas vou dormir um pouco. Use o comando ps -l para conferir o meu
estado e o do meu pai. Daqui a pouco eu acordo.

robson@noteRAS:~/codigos$ ps -l
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  1839  1829  0  80   0 -  1933 wait  pts/2      00:00:00 bash
0 R  1000  2910  1839  98  80   0 -   502 -    pts/2      00:04:40 testa_zombie_1
1 Z  1000  2911  2910  0  80   0 -    0 exit  pts/2      00:00:00 testa_zombie_1 <defunct>
0 S  1000  2941  1839  0  80   0 -   502 wait  pts/2      00:00:00 testa_zombie_3
1 S  1000  2942  2941  0  80   0 -   502 hrtim pts/2      00:00:00 testa_zombie_3
0 R  1000  2943  1839  0  80   0 -  1177 -    pts/2      00:00:00 ps
robson@noteRAS:~/codigos$ Sou eu de novo, o filho. Acordei mas vou terminar agora. Use ps -l novamente.

[2]+  Fim da execucao com status 126      ./testa_zombie_3
robson@noteRAS:~/codigos$ ps -l
F S  UID  PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  1839  1829  0  80   0 -  1933 wait  pts/2      00:00:00 bash
0 R  1000  2910  1839  98  80   0 -   502 -    pts/2      00:05:44 testa_zombie_1
1 Z  1000  2911  2910  0  80   0 -    0 exit  pts/2      00:00:00 testa_zombie_1 <defunct>
0 R  1000  2962  1839  0  80   0 -  1177 -    pts/2      00:00:00 ps
```

Exercício – Gráfico de Precedência

arvorefork.c

Exercício - Gráfico de Precedência dos Processos e enviar no AVA.

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

```
int main(void)
```

```
{
    int c2, c1;
```

```
    c2 = 0;
```

```
    c1 = fork(); /* fork number 1 */
```

```
    printf("\nFORK 1 - I am process %ld and my C2 is %d, my c1 is %d\n", (long)getpid(), c2, c1);
```

```
    if (c1 == 0)
```

```
    {
        c2 = fork(); /* fork number 2 */
```

```
        printf("\nFORK 2 - I am process %ld and my C2 is %d, my c1 is %d\n", (long)getpid(), c2, c1);
```

```
    }
    fork(); /* fork number 3 */
```

```
    printf("\nFORK 3 - I am process %ld and my C2 is %d, my c1 is %d\n", (long)getpid(), c2, c1);
```

```
    if (c2 > 0)
```

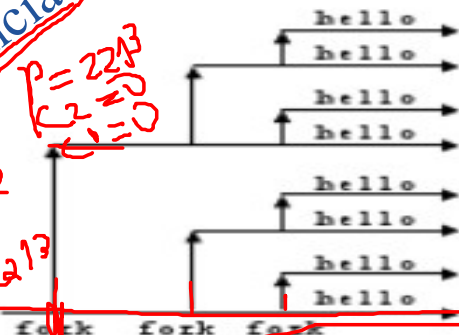
```
    {
        fork(); /* fork number 4 */
```

```
        printf("\nFORK 4 - I am process %ld and my C2 is %d, my c1 is %d\n", (long)getpid(), c2, c1);
```

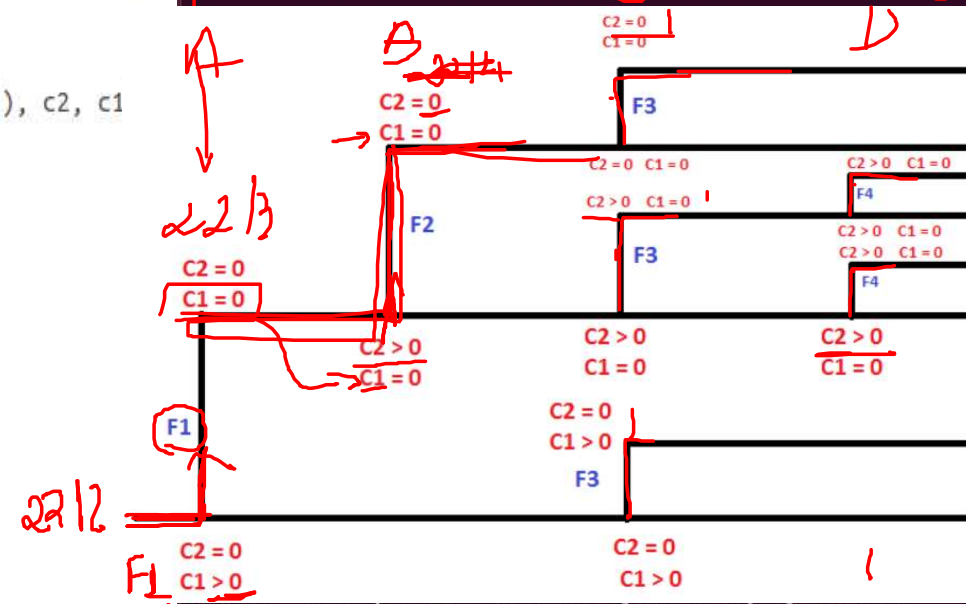
```
    }
    exit(0);
}
```

Gráfico de Precedência

P = 2212
C2 = 0
C1 = 2213



```
FORK 1 - I am process 2212 and my C2 is 0, my c1 is 2213
FORK 3 - I am process 2212 and my C2 is 0, my c1 is 2213
FORK 3 - I am process 2214 and my C2 is 0, my c1 is 2213
```



Ponteiro para sincronização

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
```

```
int *flag;
```

```
int main (void)
```

```
{
```

```
    int shmid, pid;
```

```
    /* cria um segmento de memória partilhada para um vetor flag de 3 posições */
```

```
    shmid = shmget (IPC_CREAT, 3*sizeof(int), IPC_CREAT | 0644);
```

```
    // O comando abaixo aloca um segmento/area para o vetor de 3 posições criado antes. Semelhante ao malloc
```

```
    flag = (int*) shmat (shmid, NULL, 0);
```

```
    flag[0] = 0;
```

```
    flag[1] = 0;
```

```
    flag[2] = 0; // 0 pai e 1 filho (faz o papel do turn)
```

```
    if (fork() > 0) // pai
```

```
    {
```

```
    }
```

```
    Else //filho
```

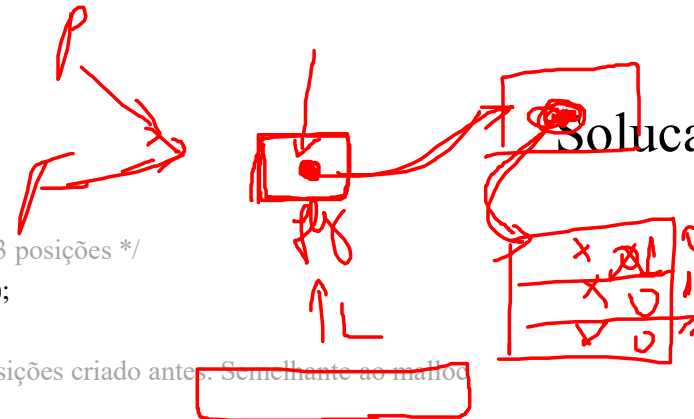
```
    {
```

```
    }
```

```
    shmdt(flag);
```

```
    shmctl(shmid, IPC_RMID, NULL);
```

```
}
```



Introdução a Processos

- **Referências Utilizadas:**

- Livro do Tanenbaum
 - Sistemas Operacionais Modernos
 - www.cs.vu.nl/~ast
- Livro do Silberschatz
 - Operating System Concepts
 - www.bell-labs.com/topic/books/aos-book/
- Livro do Machado e Maia
 - Arquitetura de Sistemas Operacionais.

