

Fundamentos de Sistemas Operacionais

Professor: Cristiano Bonato Both



JESUÍTAS BRASIL



Somos infinitas possibilidades

Sumário

- Representação de processo
 - Bloco descritor de processo
 - Modelo de processo
- Programação concorrente
- O problema do compartilhamento de recurso
- Relação produtor-consumidor
- Referência



JESUÍTAS BRASIL



UNISINOS

Somos infinitas possibilidades

Introdução

- Multiprogramação pressupõe a existência simultânea de vários processos disputando o processador
- Necessidade de “intermediar” esta disputa de forma justa
- Necessidade de “representar” um processo
 - Estrutura de dados



JESUÍTAS BRASIL



Somos infinitas possibilidades

Representação de Processo

- Processo é um programa em execução
 - Área na memória para código, dados e pilha
- Possui uma série de estados para representar sua evolução no tempo
 - Organizar os processos em diferentes estados
 - Determinar eventos que realizam a transição entre os estados
 - Determinar quando um processo tem direito a “utilizar” o processador
- Necessário manter informações a respeito do processo
 - e.g., prioridade, localização em memória, estado atual, direitos de acesso, recursos que emprega, etc.



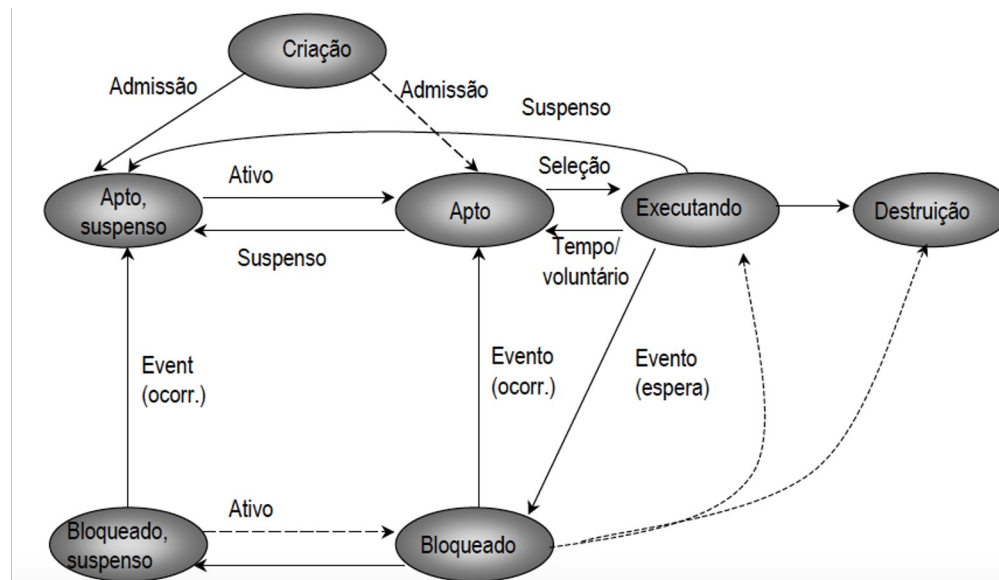
Bloco Descritor de Processo

- Abstração de processo é implementado através de uma estrutura de dados
 - Bloco descritor de processos (*Process Control Block* - PCB)
- Informações normalmente presentes em um descritor de processo
 - Prioridade
 - Localização e tamanho na memória principal
 - Identificação de arquivos abertos
 - Informações de contabilidade (tempo de CPU, etc.)
 - Estado do processador (apto, executando, etc.)
 - Contexto de execução
 - Apontadores para encadeamento dos próprios descritores



Processos e Filas

- Um processo sempre faz parte de alguma fila
- Geralmente a própria estrutura de descritores de processos são empregadas como elementos dessas filas
 - Alocação dinâmica de memória
- Eventos realizam transição de uma fila a outra



Exemplo de PCB

- Estrutura de dados representando bloco descritor de processo

```
typedef struct DescProc {  
    char    estado_atual;  
    int     prioridade;  
    unsigned inicio_memoria;  
    unsigned tamanho_memoria;  
    //struct  Arquivos arquivos_abertos[20];  
    unsigned tempo_cpu;  
    unsigned proc_pc;  
    unsigned proc_sp;  
    unsigned proc_acc;  
    unsigned proc_rx;  
    struct  DescProc *proximo;  
} DescProc;  
/* Quantidade máxima permitida de processos criados no sistema */  
DescProc tab_desc[MAX_PROCESS];
```



Exemplo de PCB

- Estrutura de filas e inicialização

```
struct desc_proc *desc_livre;  
struct desc_proc *espera_cpu;  
struct desc_proc *usando_cpu;  
struct desc_proc *bloqueados;
```

```
/* Inicialização das estruturas de Controle */
```

```
for(i = 0; i < MAX_PROCESS - 1; i++)  
    tab_desc[i].proximo = &tab_desc[i + 1];
```

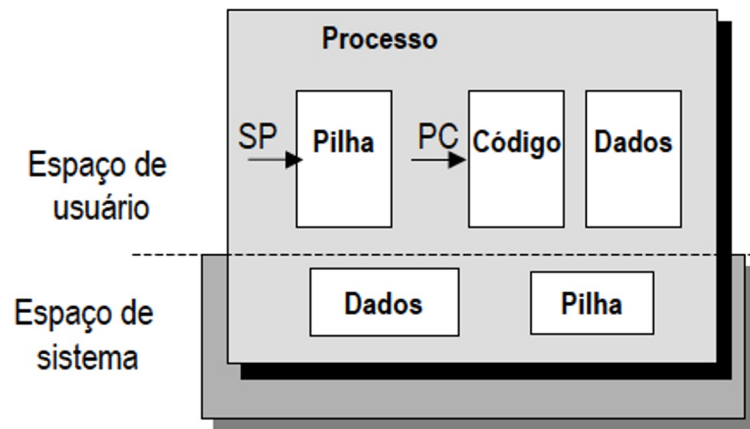
```
tab_desc[i].proximo = NULL;  
desc_livre.primeiro = &tab_desc[0];  
desc_livre.ultimo = &tab_desc[MAX_PROCESS - 1];
```

```
espera_cpu.primeiro = NULL;  
espera_cpu.ultimo = NULL;  
usando_cpu.primeiro = NULL;  
usando_cpu.ultimo = NULL;  
bloqueados.primeiro = NULL;  
bloqueados.ultimo = NULL;
```



Modelo de Processo

- Processo é representado por:
 - Espaço de endereçamento: área para armazenamento da imagem do processo
 - Estruturas internas do sistema (tabelas internas, etc.)
 - Mantidos no descritor de processos
 - Contexto de execução (pilha, programa, dados, etc.)

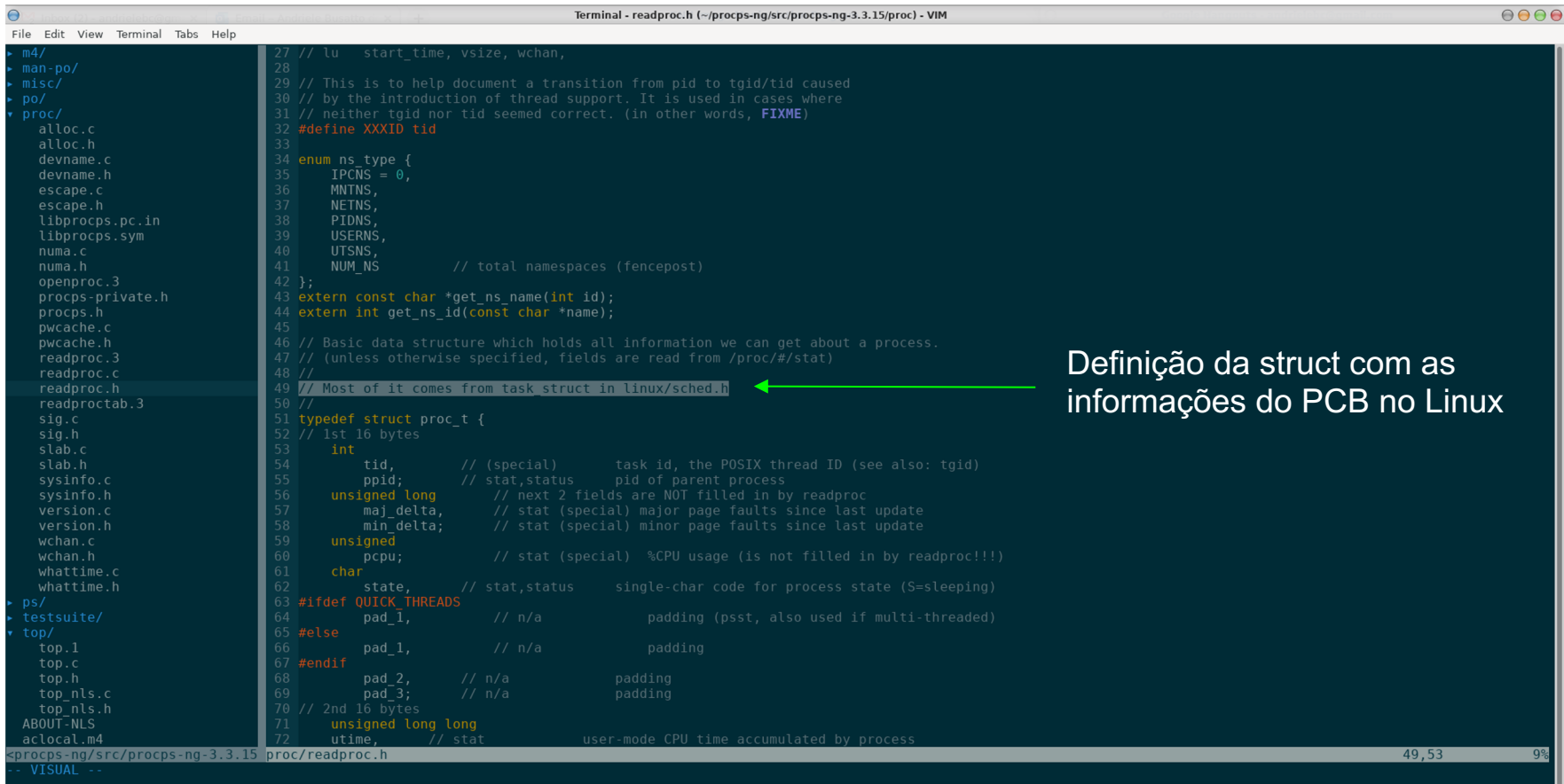


PC = *Program Counter*
SP = *Stack Pointer*



Exemplo

- Informações dos processos em execução
- No Ubuntu instalar libprocps-dev e ir em /usr/include/proc



```
Terminal - readproc.h (~/.procps-ng/src/procps-ng-3.3.15/proc) - VIM
File Edit View Terminal Tabs Help
m4/
man-po/
misc/
po/
proc/
  alloc.c
  alloc.h
  devname.c
  devname.h
  escape.c
  escape.h
  libprocps.pc.in
  libprocps.sym
  numa.c
  numa.h
  openproc.3
  procps-private.h
  procps.h
  pwcache.c
  pwcache.h
  readproc.3
  readproc.c
  readproc.h
  readproctab.3
  sig.c
  sig.h
  slab.c
  slab.h
  sysinfo.c
  sysinfo.h
  version.c
  version.h
  wchan.c
  wchan.h
  whattime.c
  whattime.h
ps/
testsuite/
top/
  top.1
  top.c
  top.h
  top_nls.c
  top_nls.h
ABOUT-NLS
aclocal.m4
27 // lu start_time, vsize, wchan,
28
29 // This is to help document a transition from pid to tgid/tid caused
30 // by the introduction of thread support. It is used in cases where
31 // neither tgid nor tid seemed correct. (in other words, FIXME)
32 #define XXXID tid
33
34 enum ns_type {
35     IPCNS = 0,
36     MNTNS,
37     NETNS,
38     PIDNS,
39     USERNS,
40     UTSNS,
41     NUM_NS          // total namespaces (fencepost)
42 };
43 extern const char *get_ns_name(int id);
44 extern int get_ns_id(const char *name);
45
46 // Basic data structure which holds all information we can get about a process.
47 // (unless otherwise specified, fields are read from /proc/#/stat)
48 //
49 // Most of it comes from task_struct in linux/sched.h
50 //
51 typedef struct proc_t {
52     // 1st 16 bytes
53     int
54         tid,          // (special) task id, the POSIX thread ID (see also: tgid)
55         ppid;         // stat,status pid of parent process
56     unsigned long
57         maj_delta,    // stat (special) major page faults since last update
58         min_delta;    // stat (special) minor page faults since last update
59     unsigned
60         pcpu;         // stat (special) %CPU usage (is not filled in by readproc!!!)
61     char
62         state,        // stat,status single-char code for process state (S=sleeping)
63 #ifdef QUICK_THREADS
64         pad_1,         // n/a padding (psst, also used if multi-threaded)
65 #else
66         pad_1,         // n/a padding
67 #endif
68         pad_2,        // n/a padding
69         pad_3;        // n/a padding
70     // 2nd 16 bytes
71     unsigned long long
72         utime,        // stat user-mode CPU time accumulated by process
73
49,53 9%
```

Definição da struct com as informações do PCB no Linux



JESUÍTAS BRASIL



Somos infinitas possibilidades

Exemplo

- No fonte do kernel: /usr/src/linux/include/linux/sched.h
- Fontes do kernel: <https://www.kernel.org/>



```
Terminal - sched.h (~/.Downloads/linux-5.2.9/include/linux) - VIM
File Edit View Terminal Tabs Help
583
584 struct task_struct {
585 #ifdef CONFIG_THREAD_INFO_IN_TASK
586 /*
587  * For reasons of header soup (see current_thread_info()), this
588  * must be the first element of task_struct.
589  */
590 struct thread_info      thread_info;
591 #endif
592 /* -1 unrunnable, 0 runnable, >0 stopped: */
593 volatile long           state;
594
595 /*
596  * This begins the randomizable portion of task_struct. Only
597  * scheduling-critical items should be added above here.
598  */
599 randomized_struct_fields_start
600
601 void                    *stack;
602 refcount_t              usage;
603 /* Per task flags (PF_*), defined further below: */
604 unsigned int             flags;
605 unsigned int             ptrace;
606
607 #ifdef CONFIG_SMP
608 struct llist_node        wake_entry;
609 int                      on_cpu;
610 #ifdef CONFIG_THREAD_INFO_IN_TASK
611 /* Current CPU: */
612 unsigned int             cpu;
613 #endif
614 unsigned int             wakee_flips;
615 unsigned long            wakee_flip_decay_ts;
616 struct task_struct       *last_wakee;
617
618 /*
619  * recent_used_cpu is initially set as the last CPU used by a task
620  * that wakes affine another task. Waker/wakee relationships can
621  * push tasks around a CPU where each wakeup moves to the next one.
622  * Tracking a recently used CPU allows a quick search for a recently
623  * used CPU that may be idle.
624  */
625 int                      recent_used_cpu;
626 int                      wake_cpu;
627 #endif
628 int                      on_rq;
629
630 include/linux/sched.h
-- VISUAL --
```



Programação Concorrente

- **Programa sequencial:** executado por apenas um processo
 - Existe apenas um fluxo de controle
- **Programa concorrente:** executado por diversos processos que cooperam entre si (ou não) para realização de uma tarefa (aplicação)
 - Existem vários fluxos de controle
 - Necessidade de interação para troca de informações (sincronização)



Programação Concorrente

- Exemplos de aplicações/termos:
 - Paralelismo real: só ocorre em máquinas multiprocessadas
 - Paralelismo “aparente” (concorrência): máquinas monoprocessadas
 - Execução simultânea versus “estado de execução” simultaneamente



JESUÍTAS BRASIL



Somos infinitas possibilidades

Programação Concorrente

- Composta por um conjunto de processos sequenciais que são executados concorrentemente
- Processos disputam recursos comuns
 - e.g., variáveis, periféricos, etc.
- Um processo pode cooperar com uma tarefa, mas é capaz de afetar, ou ser afetado, pela execução de outro processo



JESUÍTAS BRASIL



Somos infinitas possibilidades

Programação Concorrente

- Vantagens:
 - Aumento de desempenho
 - Permite a exploração do paralelismo real disponível em máquinas multiprocessadas
 - Sobreposição de operações de E/S com processamento
 - Facilidade de desenvolvimento de aplicações que possuem um paralelismo intrínseco



JESUÍTAS BRASIL



Somos infinitas possibilidades

Programação Concorrente

- Desvantagem:
 - Programação complexa
 - Aos erros “comuns” se adicionam erros próprios ao modelo
 - Diferenças de velocidade relativas de execução dos processos
 - Aspecto não-determinístico
 - Difícil depuração



JESUÍTAS BRASIL



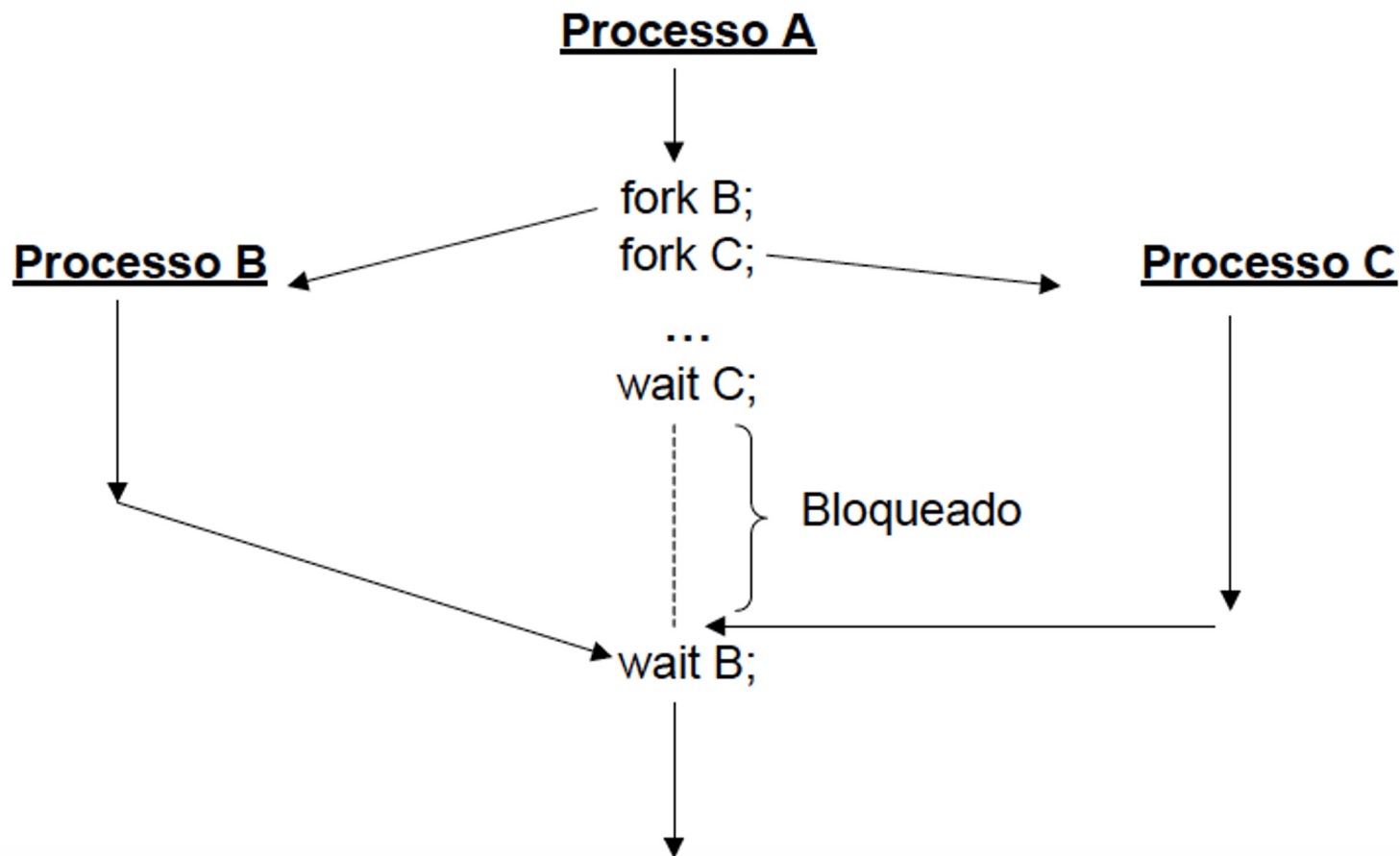
Somos infinitas possibilidades

Especificação do Paralelismo

- Necessidade de especificar o paralelismo definindo:
 - Quantos processos participarão?
 - Quem fará o que?
 - Dependência entre as tarefas (grafos)
- Notação para expressar paralelismo
 - *fork/wait*
 - Existem várias primitivas e cada linguagem possui sua forma de utilizar



Fork e wait



Comentários Gerais

- Primitivas de alto nível, para descrição do paralelismo, podem ser traduzidas por pré-compiladores e/ou interpretadores para primitivas de baixo nível
- Processos paralelos podem ser executados em qualquer ordem
 - Duas execuções consecutivas do mesmo programa, com os mesmos dados de entrada, podem gerar resultados diferentes
 - Possibilidade de “forçar” a execução em uma determinada ordem



Compartilhamento de Recursos

- A programação concorrente implica em um compartilhamento de recursos
 - Variáveis compartilhadas são recursos essenciais para a programação concorrente
- Acessos a recursos compartilhados devem ser feitos de forma a manter um estado coerente e correto do sistema



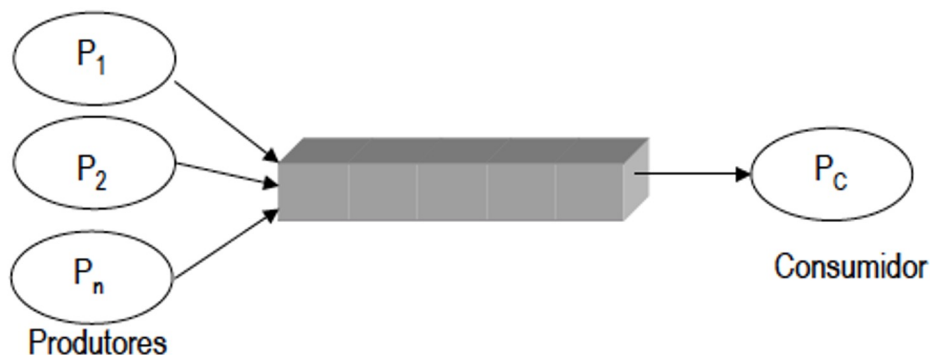
JESUÍTAS BRASIL



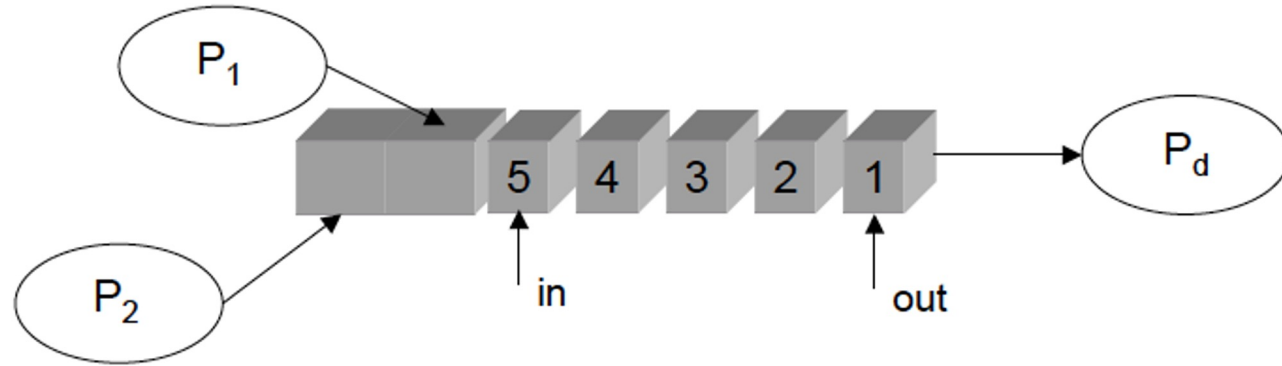
Somos infinitas possibilidades

Produtor-consumidor

- Produtor-consumidor é uma situação comum em sistemas operacionais
- Servidor de impressão:
 - Processos usuários produzem “impressões”
 - Impressões são organizadas em uma fila a partir da qual um processo (consumidor os lê e envia para a impressora



Produtor-consumidor



- Suposições:
 - Fila de impressão é um *buffer* circular
 - Existência de um ponteiro (*in*) que indica uma posição onde a impressão é inserida para aguardar o momento de ser efetivamente impressa
 - Existência de um ponteiro (*out*) que aponta para a impressão que está sendo realizada



Produtor-consumidor

Problema da Seção Crítica

- **Condição de corrida (*race condition*)**
 - Situação que ocorre quando vários processos manipulam o mesmo conjunto de dados concorrentemente e o resultado depende da ordem em que os acessos são feitos
- **Seção crítica**
 - Segmento de código no qual um processo realiza a alteração de um recurso compartilhado



Referências Bibliográficas

- SILBERSCHATZ, A.; GALVIN, Peter; GAGNE Greg, Operating System Concepts Essentials. John Wiley & Sons, Inc. 2th edition, 2013.
- TANENBAUM, Andrew S. Sistemas operacionais modernos. 3a. ed. São Paulo: Pearson, 2009-2013. p. 653.
- OLIVEIRA, Rômulo; CARÍSSIMI, Alexandre; TOSCANI, Simão. Sistemas Operacionais. Porto Alegre: Bookman, 4a. ed. 2010.



JESUÍTAS BRASIL



Somos infinitas possibilidades