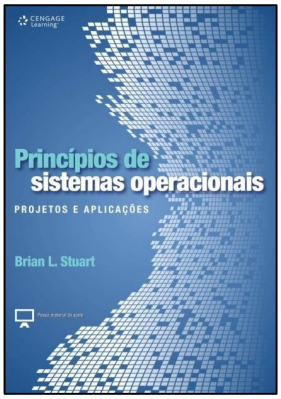
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
SISTEMAS OPERACIONAIS – Aula 02 – 1º SEMESTRE/2020
PROF. LUCIANO SILVA

TEORIA: GERENCIADOR DE PROCESSOS (PARTE I)

Nossos **objetivos** nesta aula são:

- conhecer o conceito de processo e sua representação como estrutura de dados
- conhecer o conceito de escalonamento (*scheduling*)
- conhecer o conceito de mudança de contexto



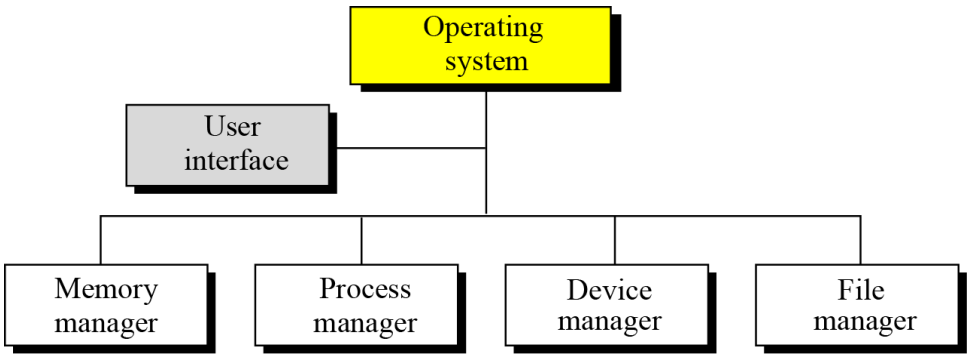
Para esta aula, usamos como referência as seções 5.1, 5.2, 5.4 (exceto algoritmos) e 5.5 do **Capítulo 5 (Princípios de gerenciamento de processos)** do nosso livro-texto:

STUART, B.L., **Princípios de Sistemas Operacionais: Projetos e Aplicações**. São Paulo: Cengage Learning, 2011.

Não deixem de ler estas seções depois desta aula!

PROCESSOS

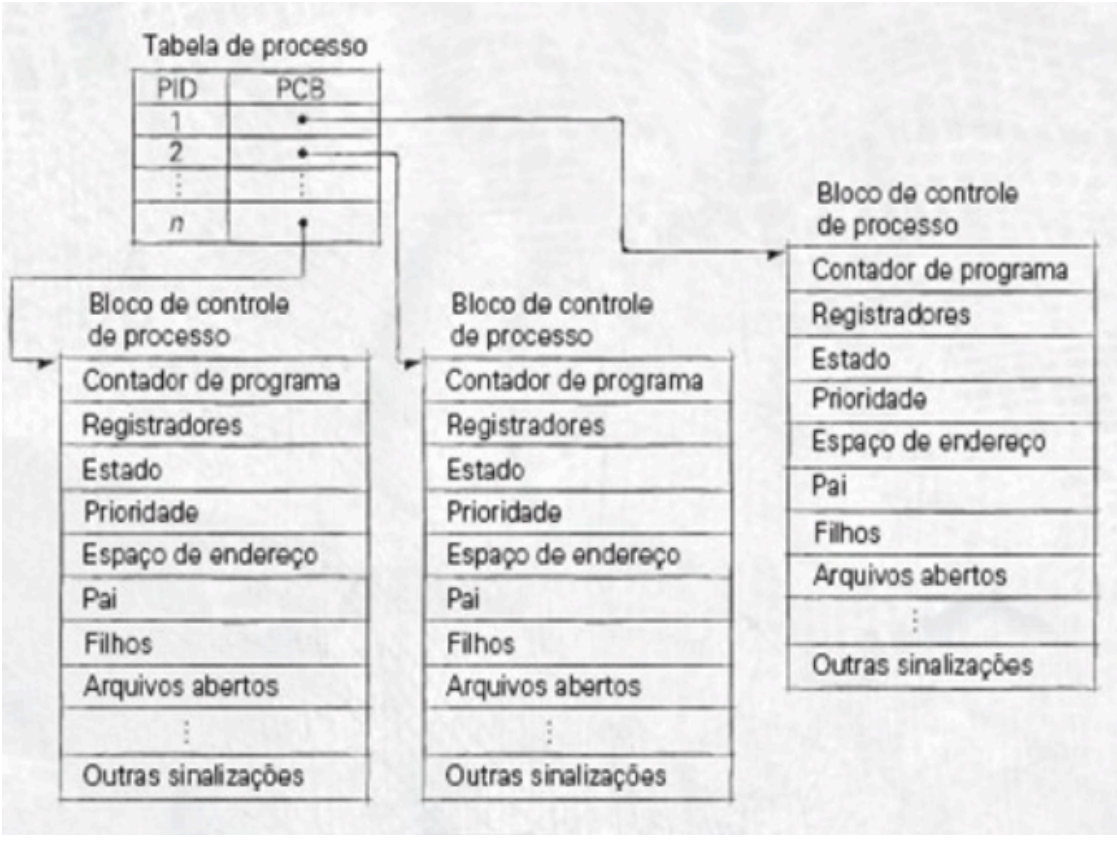
- O núcleo (kernel) de um sistema operacional, geralmente, é formado por quatro módulos de gerenciamento (managers):



- Cada um destes gerenciadores é responsável por um tipo de objeto: **gerenciador de memória** (memória), **gerenciador de processos** (programas em execução), **gerenciador de dispositivos** (teclados, vídeo, discos,...) e **gerenciador de arquivos** (dados e pastas armazenados em memória secundária como, por exemplo, arquivos em discos).

- Um **processo** é um **programa em execução**. Para controlar a execução, o S.O. utiliza uma estrutura de dados para representar o processo chamada **BCP (Bloco de Controle de Processo)**. A estrutura típica de um BCP é mostrada abaixo:

- Cada uma destas estruturas possui uma entrada na Tabela de Processos, que contém todos os processos sob o controle de um S.O.. Esta tabela é indexada pelo identificador do processo (PID) e, associado a cada PID, existe um BCP.



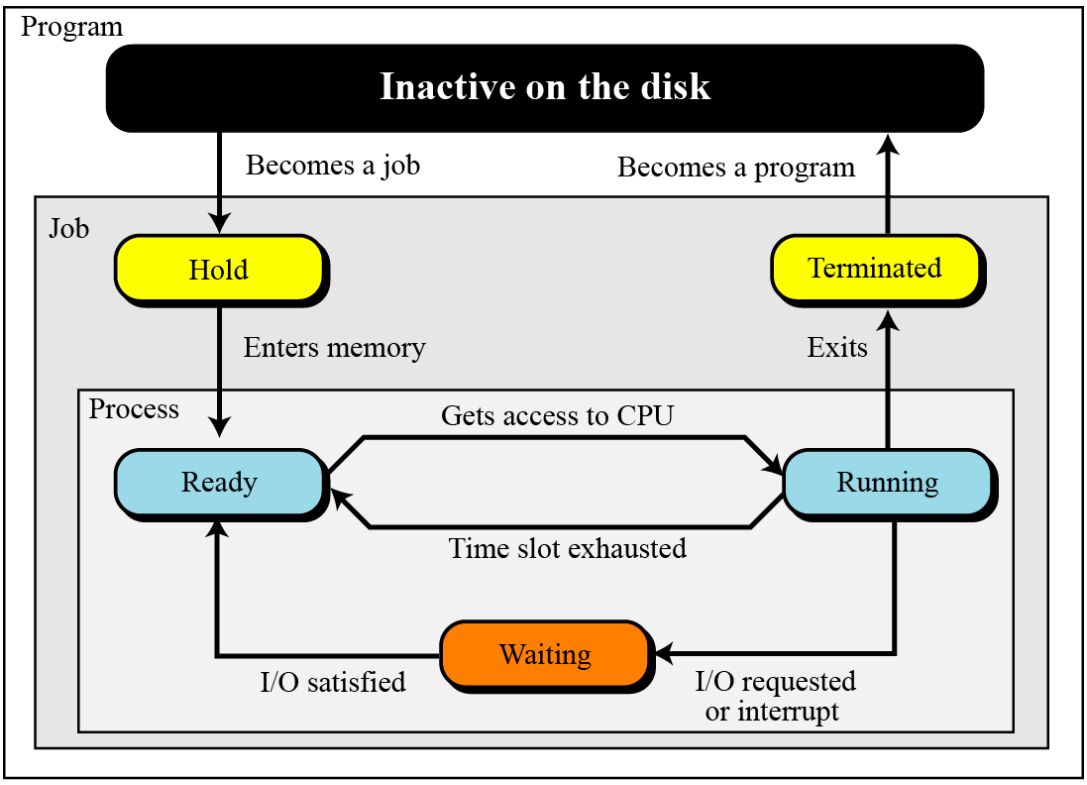
EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Abaixo, temos parte do arquivo proc.h, que o o MINIX utiliza como representação de processo. Identifique os campos que representam um processo no MINIX.

```
struct proc {
  struct stackframe_s p_reg;   /* process' registers saved in stack frame */

#if (CHIP == INTEL)
  reg_t p_ldt_sel;             /* selector in gdt giving ldt base and limit*/
  struct segdesc_s p_ldt[4];   /* local descriptors for code and data */
                               /* 4 is LDT_SIZE - avoid include protect.h */
#endif /* (CHIP == INTEL) */

#if (CHIP == M68000)
  reg_t p_splow;               /* lowest observed stack value */
  int p_trap;                  /* trap type (only low byte) */
  char *p_crp;                 /* mmu table pointer (really struct _rpr *) */
  int p_nflips;                /* statistics */
#if defined(FPP)
  struct fsave p_fsave;        /* FPP state frame and registers */
  int align2;                  /* make the struct size a multiple of 4 */
#endif
#endif /* (CHIP == M68000) */

  reg_t *p_stguard;            /* stack guard word */

  int p_nr;                    /* number of this process (for fast access) */

  char p_int_blocked;          /* nonzero if int msg blocked by busy task */
  char p_int_held;             /* nonzero if int msg held by busy syscall */
  struct proc *p_nextheld;     /* next in chain of held-up int processes */

  int p_flags;                 /* SENDING, RECEIVING, etc. */
  struct mem_map p_map[NR_SEGS];/* memory map */
  pid_t p_pid;                 /* process id passed in from MM */
  int p_priority;              /* task, server, or user process */

  clock_t user_time;           /* user time in ticks */
  clock_t sys_time;            /* sys time in ticks */
  clock_t child_utime;         /* cumulative user time of children */
  clock_t child_stime;         /* cumulative sys time of children */

  timer_t *p_exptimers;                /* list of expired timers */

  …
};
```

GERENCIADOR DE PROCESSOS

- Um gerenciador de processos controla a execução de programas. Para se transformar em processo, o programa para por uma fase intermediária chamada **tarefa (job)**, quando ele deve ser preparado para executar (carga da memória) ou terminar a execução .

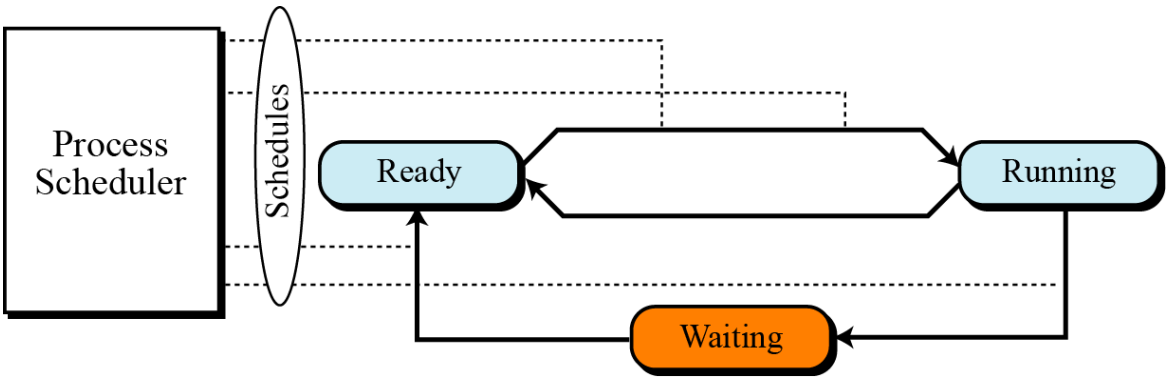


- Um processo pode estar em um dos seguintes estados:
  - **pronto (ready):** está aguardando a CPU estar livre para ser executado
  - **executando ou rodando (running):** está efetivamente usando a CPU
  - **em espera (waiting):** está esperando por algum dispositivo de E/S (exemplo, esperando um usuário digitar um número no teclado)

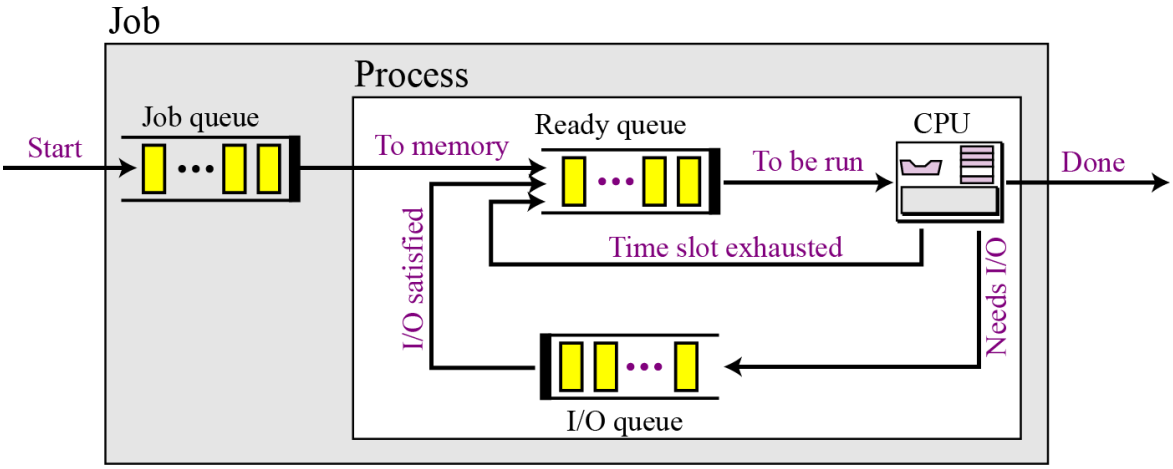
- Um processo, quando ganha a CPU para execução, possui um tempo máximo de execução chamado **fatia de tempo (time slice)**. A fatia de tempo varia, geralmente, de 10 ms a 100 ms em vários SOs.

- Quando termina a fatia de tempo do processo, ele precisa deixar a CPU para outro processo possa usá-la.

- A decisão de qual processo irá usar a CPU depende do **escalonador (scheduler)**. Existem diversas técnicas de escalonamento: FCFS, Round Robin, Prioridades,... . Veremos, em aulas posteriores, estes algoritmos em detalhes.



- O controle de processos prontos e em espera é feito através de filas de processos.

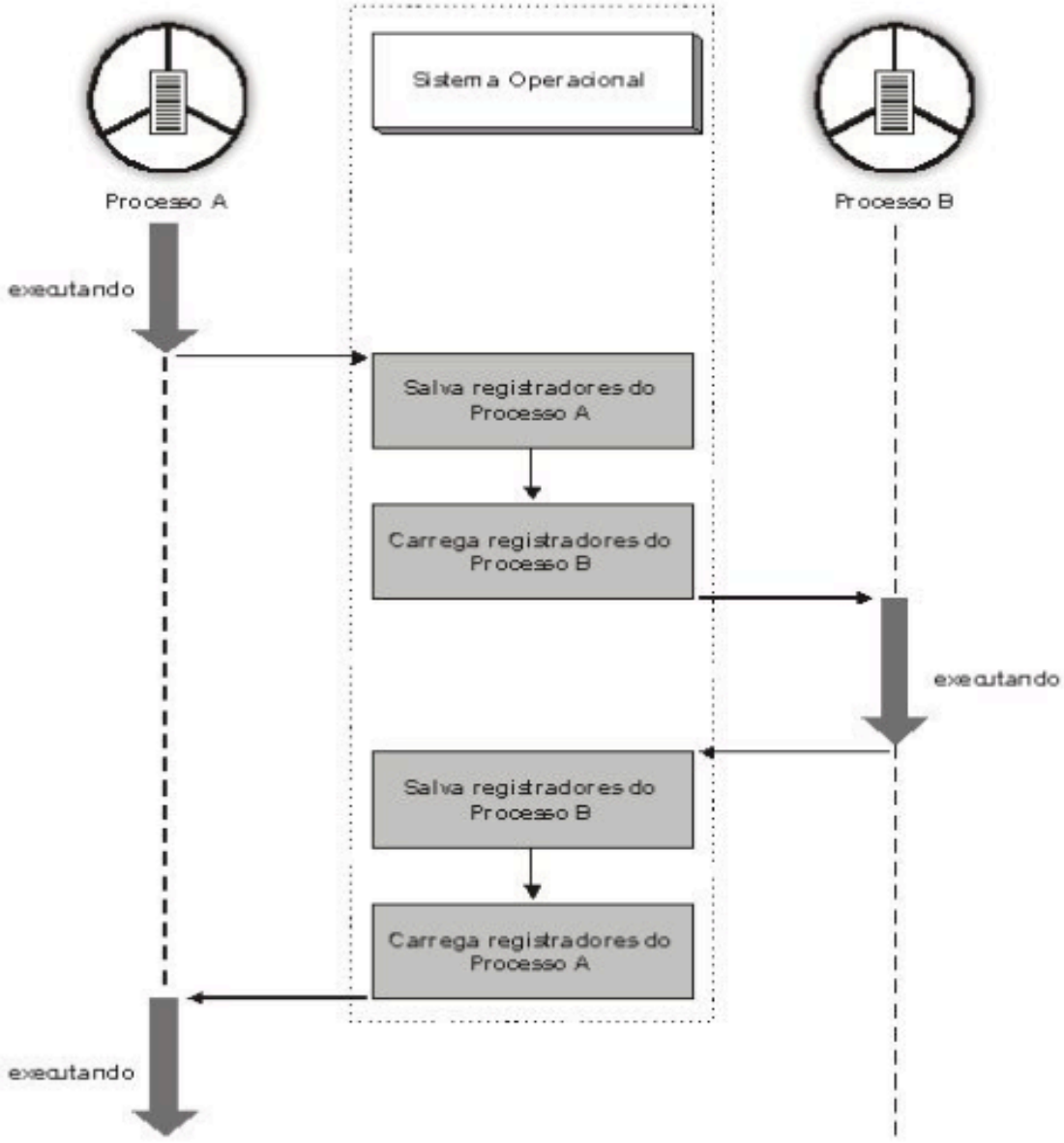


EXERCÍCIO COM DISCUSSÃO EM DUPLAS

Na imagem acima, temos uma CPU com um único core. Como você modificaria a estrutura do núcleo do seu S.O. se a sua CPU tivesse mais que um core de processamento (por exemplo, um quadcore).

MUDANÇA DE CONTEXTO (*CONTEXT SWITCHING*)

- Quando um **processo muda de estado** (PRONTO, EXECUÇÃO ou EM ESPERA), dizemos que ocorreu uma **mudança de contexto (context switching)**. O esquema típico de mudança de contexto é mostrado abaixo:



- Como a **mudança de contexto** é um procedimento simples, ela pode ser **efetuada por software** pelo próprio S.O. ou **pela CPU**, para aumentar o desempenho do S.O..

EXERCÍCIO EXTRA-CLASSE

Descrever como é o escalonamento de processos nos sistemas operacionais WINDOWS, LINUX, MacOS, Android, iOS e MINIX.