Processos

- Conceituação
- Gerência de processos
- Estrutura de processos
- Implementação de processos

Multiprogramação

- "*pseudoparalelismo*": coleção de processos sendo executados alternadamente na CPU
    - **execução concorrente**
- ideia: reduzir o desperdício de CPU devido às op. de E/S
- objetivos:
    - aumentar a taxa de uso da CPU
    - melhorar utilização dos recursos
    - aumentar *throughput*

Conceito(s) de Processo

- job = processo
- programa em execução
  - "*o processo é uma diferenciação entre o programa e sua execução*"
- entidade ativa que compete por recursos oferecidos pelo SO:
  - acesso a discos, periféricos e principalmente CPU
- interage com outros processos
- execução: sequencial

Programa x Processo

- Programa
  - Entidade estática e permanente
    - Sequência finita de instruções
    - Passivo sob o ponto de vista do SO (não se altera c/ passar do tempo)
    - Armazenado em disco
- Processo
  - Entidade dinâmica e efêmera
    - Altera seu estado a medida que avança sua execução
  - Um programa pode ter várias instâncias em execução
    - diferentes processos

Processo: conceitos relacionados

- Programa – código (estático)
- Job – programa batch em execução
- Tarefa (task) – unidade atômica de computação
- Thread – processo leve (compartilha código)

Processos do SO

- Processos executam:
  - programas de usuários
  - **programas do SO**
    - *daemons* (execução em *background*)
    - estão sempre disponíveis
    - realizam serviços do SO
    - ex.:
      - *Spooler* de impressão
      - Servidor *http*
      - Servidor *ftp*

Representação da imagem de um processo

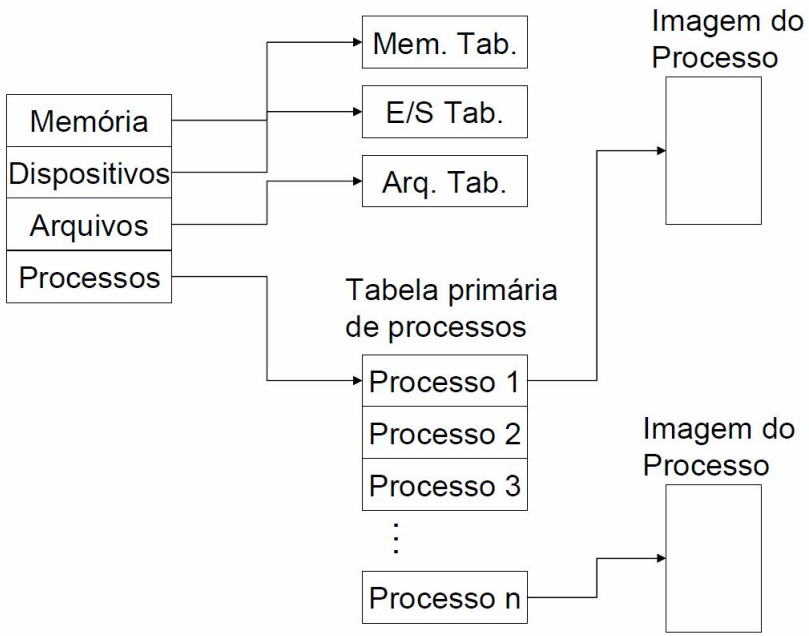
- um processo é representado por uma "imagem":
  - segmento de código (*o que ele vai fazer ?*)
  - espaço de endereçamento (*onde, na memória, ele vai fazer alguma coisa ?)*
  - contexto (*o que ele precisa para fazer alguma coisa ?*)
- parte da imagem é de responsabilidade do usuário, a outra é gerenciada em modo protegido (SO)

Componentes de um processo

- Processo inclui:
  - contador de programa (PC)
    - indica próxima instrução a executar
  - pilha de execução (stack)
    - com valores temporários (parâmetros de funções, endereços de retorno, ...)
  - área de dados
    - com os valores das variáveis globais

Implementação de processos

- Tabelas do sistema
  - SO precisa manter informações sobre o estado atual de cada processo e recurso
  - Usa estruturas de controle:
    - Tabelas de memória
    - Tabelas de E/S
    - Tabelas de arquivos
    - **Tabelas de processos**



Gerência de processos – PCB

- SO mantém uma **Tabela de Processos**
  - armazena informações que variam de um processo para outro
  - há uma entrada na tabela para cada processo
  - cada entrada é chamada de PCB (Bloco de Controle de Processos)
  - PCB
    - vetor ou lista encadeada de estruturas
    - um para cada processo do sistema

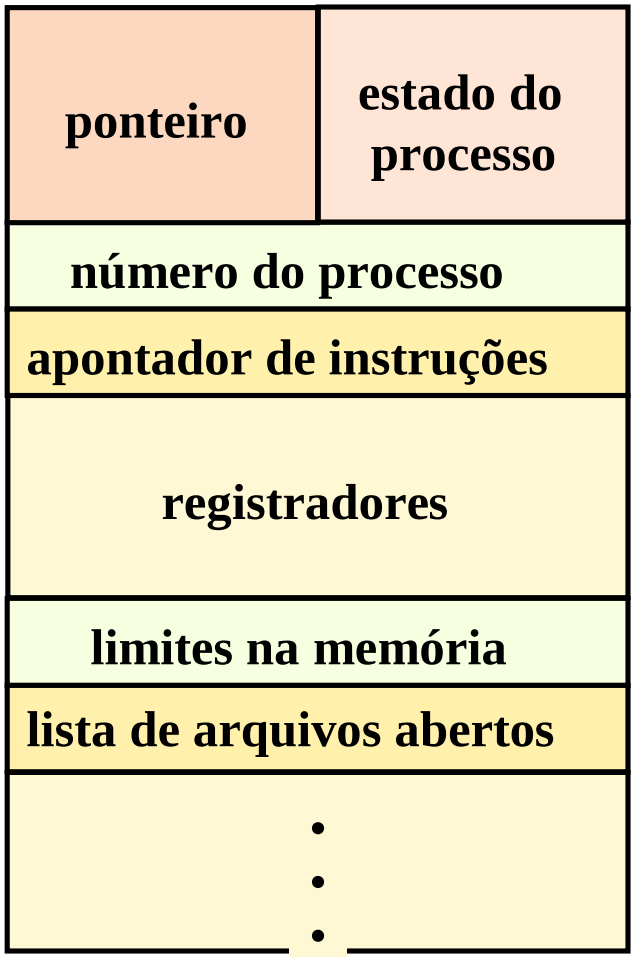
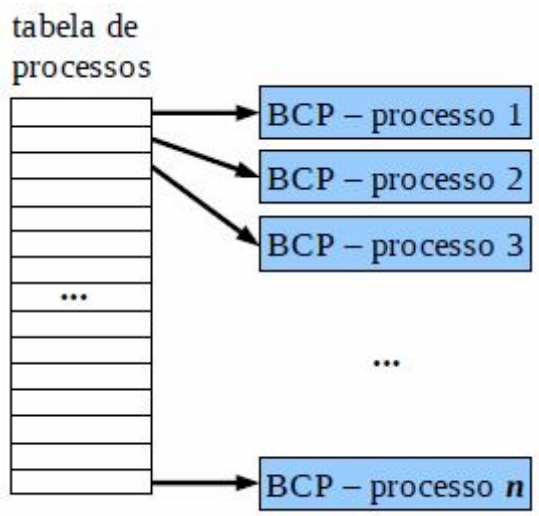
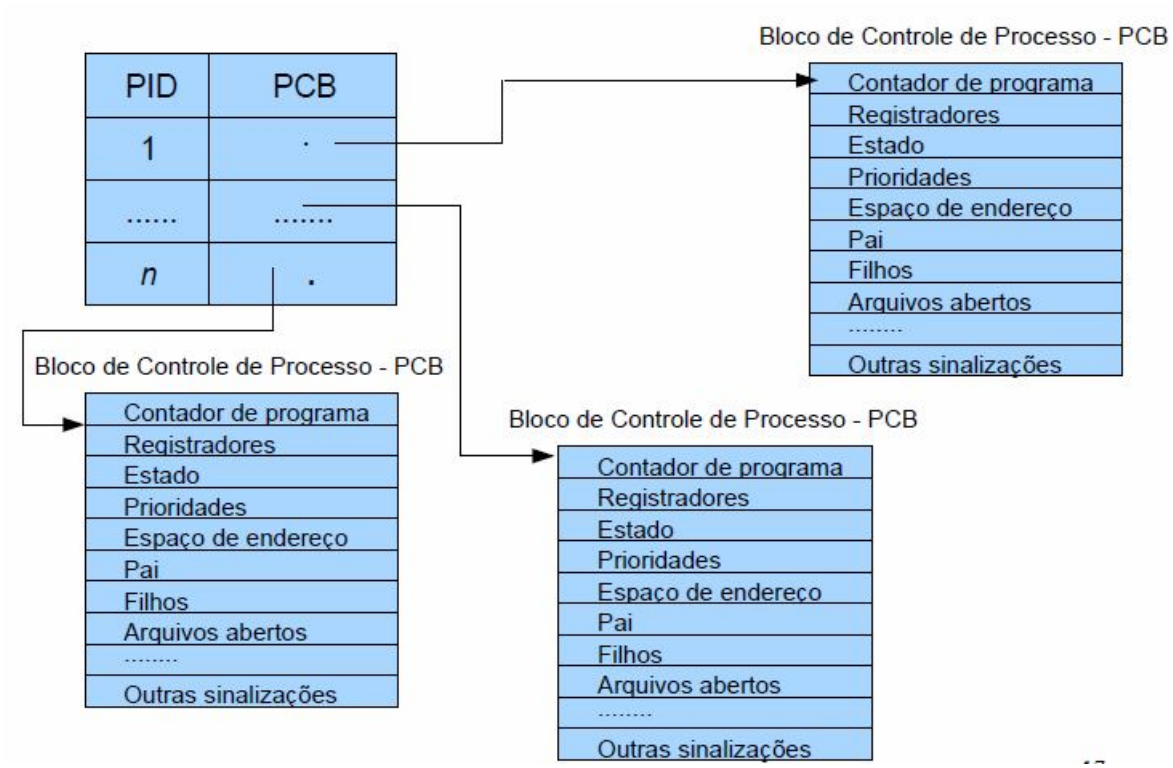
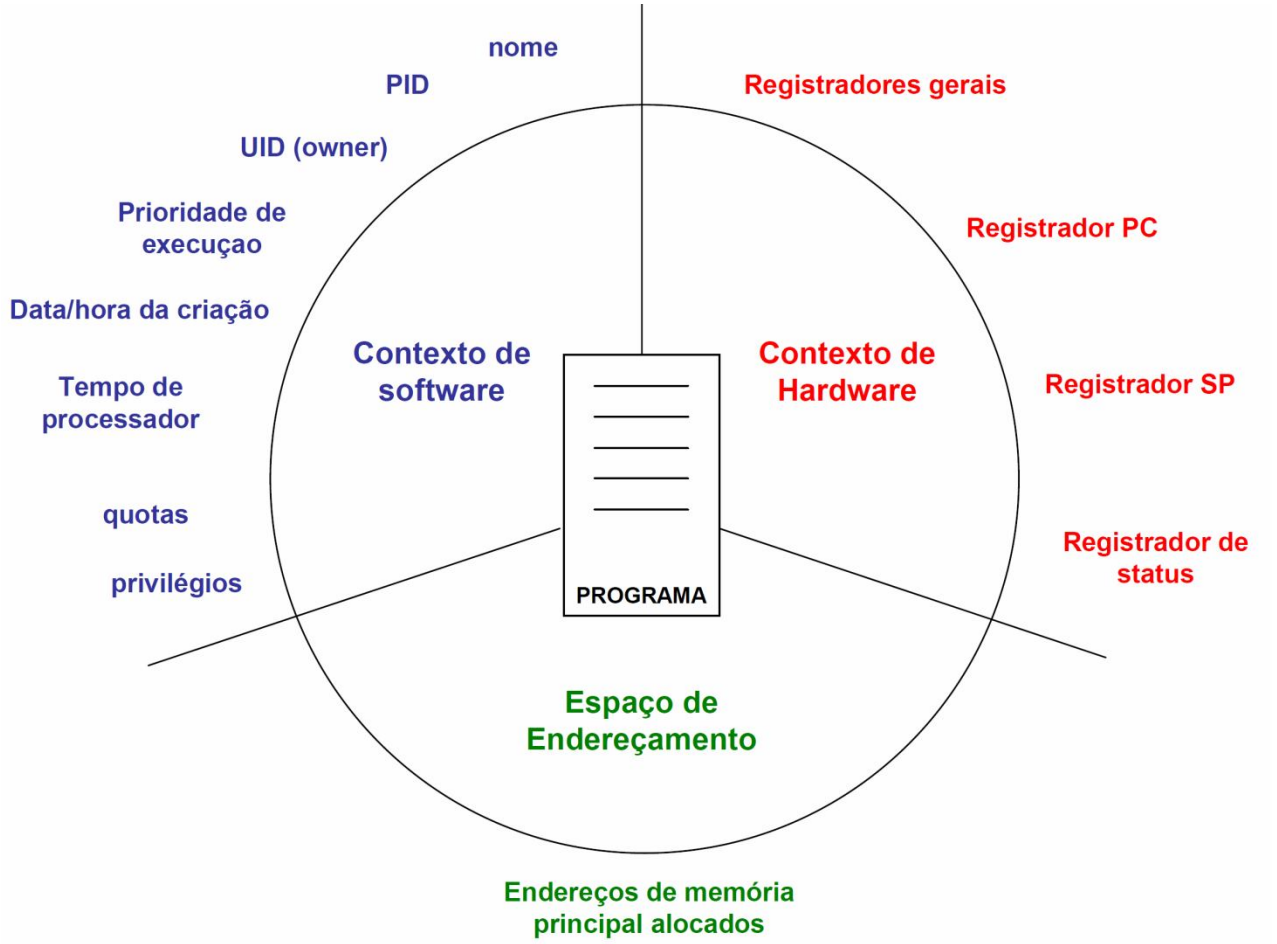


Tabela de Processos e Bloco de Controle de Processos



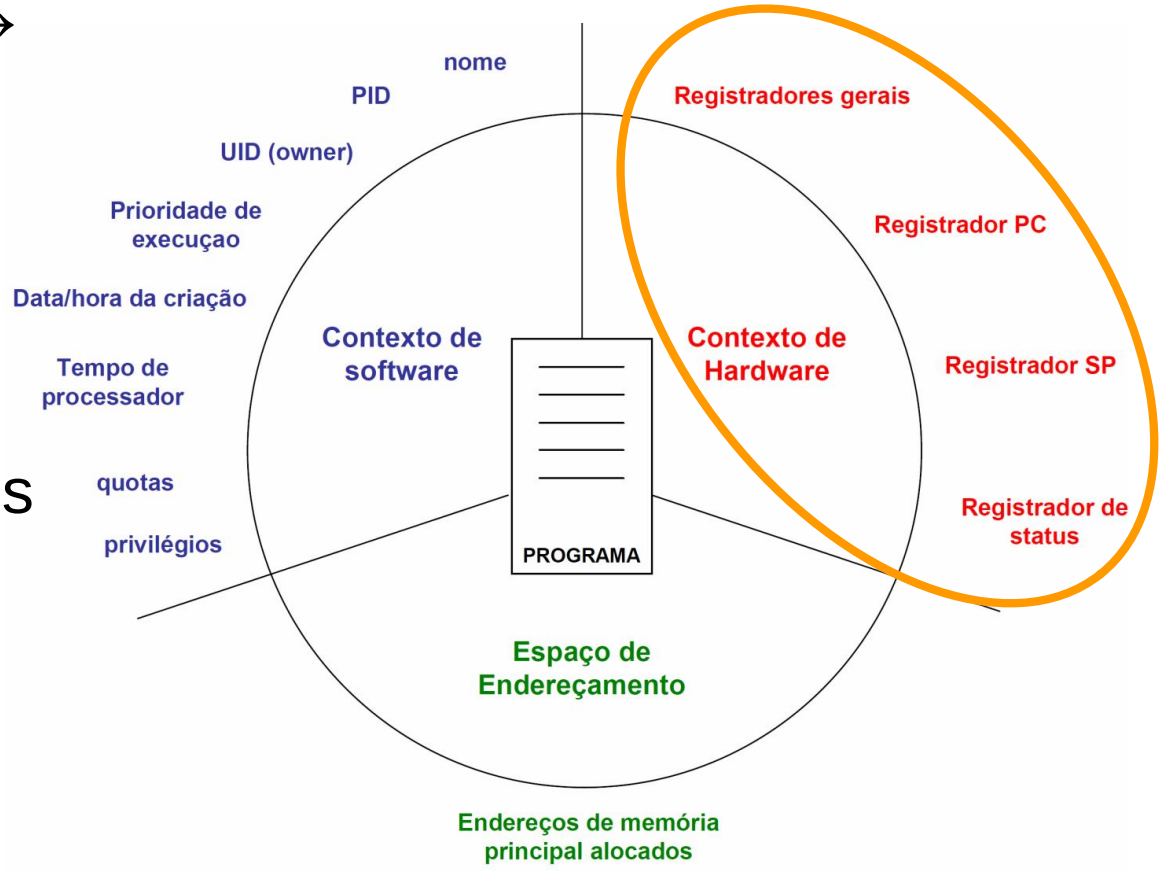


Estrutura do processo: PCB (1)



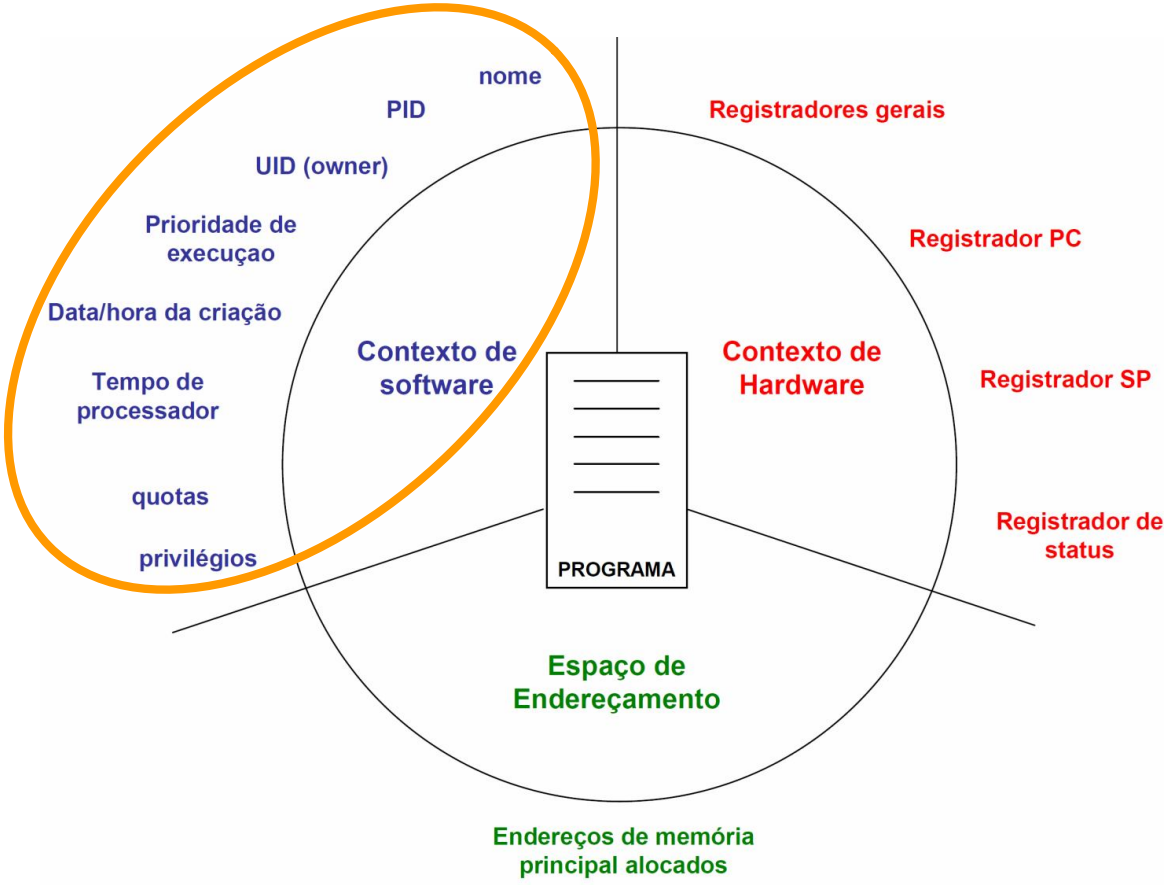
Estrutura do processo: contexto de hw (2)

- processo executando → contexto de hw armazenado nos registradores da CPU (PC, SP, PSW)
- processo perde a CPU → SO salva informações no contexto de hw
- fundamental p/ implementação de SO time-sharing



Estrutura do processo: contexto de sw (3)

- 3 grupos de informações:
  - Identificação
    - PID – id
    - UID – criador
    - GID – grupo
  - quotas
    - Ex.: máx. de arquivos abertos, máx. de memória que pode alocar, ...
  - privilégios
    - o que pode/não fazer
- determinadas na criação do processo
- dinâmicas



Estrutura do processo: espaço de endereçamento (4)

- área de memória onde reside o processo
  - onde o programa será executado
- espaço de memória p/ dados do programa
- deve ser protegido do acesso pelos demais processos

Exemplo de PCB

```
struct desc_proc {
    char estado_atual;              /* Estado do processo */
    int prioridade;                 /* Prioridade do processo */
    unsigned inicio_memoria;        /* Endereço inicial da memória */
    unsigned tamanho_memoria;       /* memória usada (bytes) */
    struct arquivo arq_abertos[20]  /* Arquivos abertos */
    unsigned tempo_de_CPU;          /* Tempo de CPU */
    unsigned proc_pc;               /* Valor do PC (registrador) */
    unsigned proc_sp;               /* Valor do SP (registrador) */
    unsigned proc_acc;              /* Valor do ACC (registrador) */
    unsigned proc_rx;               /* Valor do RX (registrador) */
    struct desc_proc *proximo       /* Aponta para o próximo */
}
struct desc_proc tab_desc[MAX_PROCESS];
```

Implementação de Processos

- Criação de processo no Windows
- Criação de processo no Unix
- Término
- Hierarquia

Criação de processos

- Sistemas antigos: apenas o SO podia criar novos processos
- Sistemas atuais: usuários podem criar e destruir processos dinamicamente
  - SO fornece chamadas para manipulação e gerência de processos

Quando criar um processo dinamicamente?

- Em sistemas de uso geral, há, basicamente, 4 meios pelos quais os processos são criados:
  - inicialização do sistema
  - execução de uma chamada de sistema p/ criação de processo, realizada por um processo em execução
  - requisição do usuário p/ criar um novo processo
  - execução de uma tarefa (job) em lote

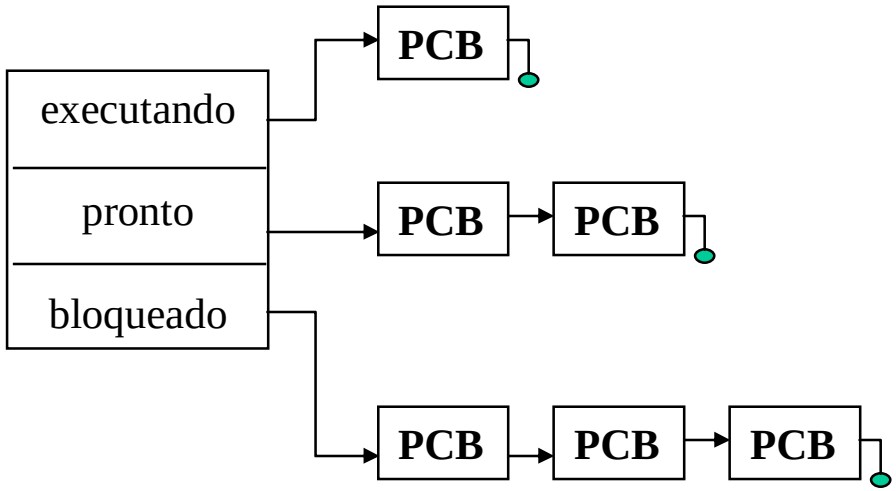
- Obs.: todo novo processo é criado por um processo existente, via syscall

Criação de processo

- Tanto no Unix, como no Windows, o novo processo possui um espaço de endereçamento diferente do processo pai (criador)
  - Não há compartilhamento de memória
  - Esta decisão de projeto mantém a consistência do sistema como um todo
- Um processo filho pode compartilhar com o pai recursos do tipo arquivos abertos, ...

Etapas da criação de um processo

- atribui um *pid* único
- aloca estruturas de dados associadas ao processo
- alocar espaço necessário em memória
- inicia PCB
  - PCB recebe as informações básicas
  - PCB é colocado na fila de prontos
- atualiza das listas do SO para manter consistência



Criação de processo – Windows

- Chamada de sistema *CreateProcess()*
  - uma única função trata tanto do processo de criação quanto da carga do novo programa no novo processo
  - possui diversos parâmetros:
    - Atributos de segurança
    - Bits que controlam se os arquivos abertos são herdados
    - Informações sobre prioridade
    - Especificação de janela (se for o caso)
    - ...

Criação de processo – Windows: Chamada de sistema *CreateProcess()*

```
#include <windows.h>
BOOL WINAPI CreateProcess(
    _in_opt     LPCTSTR lpApplicationName,
    _inout_opt  LPTSTR lpCommandLine,
    _in_opt     LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _in_opt     LPSECURITY_ATTRIBUTES lpThreadAttributes
    _in         BOOL bInheritHandles,
    _in         DWORD dwCreationFlags,
    _in_opt     LPVOID lpEnvironment,
    _in_opt     LPCTSTR lpCurrentDirectory,
    _in         LPSTARTUPINFO lpStartupInfo,
    _out        LPPROCESS_INFORMATION lpProcessInformation
);
```

Retorna:

- 0 - em caso de falha
- valor diferente de zero - em caso de sucesso

Criação de processo – Ex. Windows

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

int main(){
   STARTUPINFO si;
   PROCESS_INFORMATION p;

   ZeroMemory(&si, sizeof(si));   // varias estrut. no Windows possuem a
   si.cb = sizeof(si);            // variavel cb, que deve ser preenchida com o
   ZeroMemory(&p, sizeof(p));     // sizeof da estrut. antes do CreateProcess

   if(!CreateProcess(NULL, "C://WINDOWS/NOTEPAD.EXE", NULL, NULL, FALSE, 0, NULL, NULL, &si, &p)){
            printf( "Erro em CreateProcess: 0x%X", GetLastError() );
            return(0);
   }
   else
            printf("Processo criado com sucesso !!!\n\n\n");

   CloseHandle(p.hProcess);   // fecha handle do processo terminado
   CloseHandle(p.hThread);    // fecha handle da thread terminada

   system("pause");
}
```