Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 02 - Processos

Aula de Hoje

Aula de Hoje

Processos

Processos são uma das abstrações mais fundamentais de um S.O.:

- Um Programa Rodando

Por si só, um processo é inanimado.

- Apenas um conjunto de instruções, bytes parados no disco.

É o S.O. que pega esta sequência de bytes e a executa, transformando o seu executável em algo útil (se ele funcionar).

Processos

Em sistemas modernos, é normal querer que mais de um programa rode ao mesmo tempo.

Facilita o uso do Sistema, pois não precisamos nos preocupar se a CPU está disponível antes de rodar um programa.
Basta rodar os programas que se deseja.

A pergunta é: Apesar de só haver uma (ou poucas) CPUs, como o S.O. gera a ilusão de que temos uma quantidade praticamente infinita?

Virtualização de CPU

A resposta é através da virtualização da CPU.

Rodando um processo, parando, rodando outro, parando tão rápido que parecem que estão rodando ao mesmo tempo.

A esse processo se dá o nome de **time sharing**.

Processos são rodados concorrentemente, quantos o usuário quiser.

- Com possíveis custos de performance do processador.

Preempção

Para poder haver o conceito de vários programas rodando "ao mesmo tempo"(**pseudo-paralelismo**), o S.O. tem que, após um certo tempo de execução de um processo, retirar ele da CPU para dar espaço para outros processos poderem usar.

Este processo se chama **Preempção**.

O tempo que um processo passa de posse da CPU chama-se **Quantum**.

Mecanismos e Políticas

Mecanismos: São protocolos, hardware e códigos que implementam uma funcionalidade.

- Ex: Troca de Contexto.

Políticas: Algoritmos para tomada de decisão do S.O.

- Ex: Escalonamento, que pode usar informações históricas, de carga de trabalho e métricas de performance.

É prática comum nos S.O.s moderno separar políticas de alto nível de mecânicas de baixo nível. Modularidade.

A Abstração Processo

Chamamos a abstração de um programa sendo executado pelo S.O. de processo.

O sistema operacional guarda, para cada processo, informações essenciais para o seu funcionamento. O **estado** do processo.

- Memória do Processo, com as instruções e os dados. A memória que um processo acessa tem o nome de espaço de endereçamento.
- Registradores, incluindo o Program Counter (PC), e os frame e stack pointers.
- Lista de dispositivos de E/S em uso pelo processo.

API dos Processos

Qualquer S.O. moderno vai ter uma variação destas funções básicas em sua API para os processos

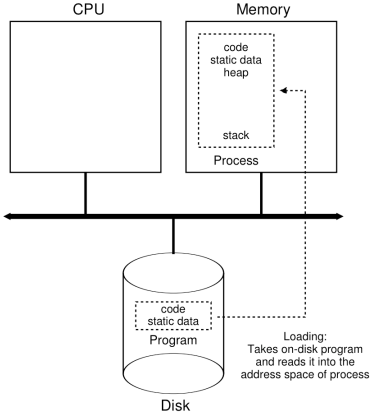
- Criar.
- Destruir.
- Esperar.
- Status.
- Outros Controles.

Criação de Processos

Primeiro Passo é carregar o código, em formato executável, de algum armazenamento secundário para a memória.

Sistemas Operacionais antigos carregavam todo o código de uma vez.

Sistemas mais modernos carregam de forma mais relaxada, carregando somente os pedaços necessários inicialmente e o resto sob demanda.



Criação de Processos

Após o carregamento dos dados estáticos, o S.O. tem outras atribuições antes de começar a executar. Ele também deve:

- **Alocar a pilha de execução.**
  - Programas normalmente alocam variáveis locais, parametros de função e endereços de retorno na pilha.
  - Também é inicializada com os parâmetros da função `main` (`argc` e `argv`).

Criação de Processos

Após o carregamento dos dados estáticos, o S.O. tem outras atribuições antes de começar a executar. Ele também deve:

- **Alocar a pilha de execução.**
  - Programas normalmente alocam variáveis locais, parametros de função e endereços de retorno na pilha.
  - Também é inicializada com os parâmetros da função `main` (`argc` e `argv`).
- **Inicializar o Heap.**
  - O Heap é onde são alocadas as variáveis geradas **dinamicamente**, ou seja, via malloc. O Heap normalmente inicia vazio, crescendo conforme os malloc são feitos.

Criação de Processos

Após o carregamento dos dados estáticos, o S.O. tem outras atribuições antes de começar a executar. Ele também deve:

- **Alocar a pilha de execução.**
  - Programas normalmente alocam variáveis locais, parametros de função e endereços de retorno na pilha.
  - Também é inicializada com os parâmetros da função `main` (`argc` e `argv`).
- **Inicializar o Heap.**
  - O Heap é onde são alocadas as variáveis geradas **dinamicamente**, ou seja, via malloc. O Heap normalmente inicia vazio, crescendo conforme os malloc são feitos.
- **Preparar entrada e saída.**
  - Por padrão, em S.O.s baseados em Unix iniciam com 3 descritores de e/s abertos: entrada padrão, saída padrão e erro.

Criação de Processos

Após o carregamento dos dados estáticos, o S.O. tem outras atribuições antes de começar a executar. Ele também deve:

- **Alocar a pilha de execução.**
  - Programas normalmente alocam variáveis locais, parametros de função e endereços de retorno na pilha.
  - Também é inicializada com os parâmetros da função `main` (`argc` e `argv`).
- **Inicializar o Heap.**
  - O Heap é onde são alocadas as variáveis geradas **dinamicamente**, ou seja, via malloc. O Heap normalmente inicia vazio, crescendo conforme os malloc são feitos.
- **Preparar entrada e saída.**
  - Por padrão, em S.O.s baseados em Unix iniciam com 3 descritores de e/s abertos: entrada padrão, saída padrão e erro.

Após tudo isto, o S.O. pode finalmente iniciar a execução do programa, indo para a primeira instrução da `main()`.

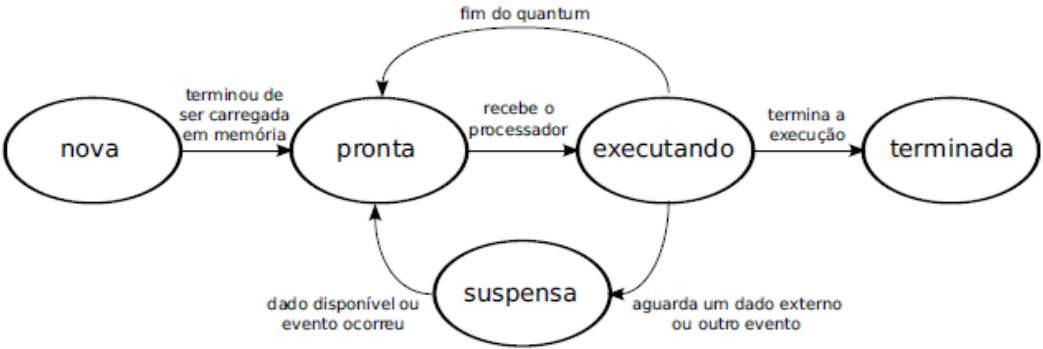
Aula de Hoje

Estados de um Processo

A grosso modo, durante a sua execução, um processo pode estar em um dos seguintes três estados.

- Executando. Estado em que o processo está quando suas instruções estão sendo executadas na CPU.
- Pronto. Estado em que o processo está pronto para ser executado, esperando apenas que o S.O. decida que é a vez dele de ser executado.
- Bloqueado. O processo efetuou alguma alguma operação em que ele deve esperar outro evento para voltar a ficar pronto para execução, como por exemplo quando é efetuada uma operação de entrada/saída.

Transições entre estados



Transições entre estados

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	
4	Running	Ready	Process_0 now done
5	–	Running	
6	–	Running	
7	–	Running	
8	–	Running	Process_1 now done

Transições entre estados

Time	Process_0	Process_1	Notes
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	Process_0 initiates I/O
4	Blocked	Running	Process_0 is blocked,
5	Blocked	Running	so Process_1 runs
6	Blocked	Running	
7	Ready	Running	I/O done
8	Ready	Running	Process_1 now done
9	Running	–	
10	Running	–	Process_0 now done

Aula de Hoje

Estruturas de Dados de um processo

S.O. é um programa, então precisa de estruturas de dados para guardar informações.

Precisa guardar não apenas as informações sobre um processo mas também as listas de todos os processos prontos e os esperando por E/S.

Para saber, por exemplo, qual processo acordar quando chegam dados de uma leitura.

A estrutura de dados que guarda os dados de um processo é conhecida como **Process Control Block (PCB)**.

Estrutura de um Processo

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
  int eip;
  int esp;
  int ebx;
  int ecx;
  int edx;
  int esi;
  int edi;
  int ebp;
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };
```

Estrutura de um Processo

```
struct proc {
  char *mem;                  // Start of process memory
  uint sz;                    // Size of process memory
  char *kstack;               // Bottom of kernel stack
                              // for this process
  enum proc_state state;      // Process state
  int pid;                    // Process ID
  struct proc *parent;        // Parent process
  void *chan;                 // If non-zero, sleeping on chan
  int killed;                 // If non-zero, have been killed
  struct file *ofile[NOFILE]; // Open files
  struct inode *cwd;          // Current directory
  struct context context;     // Switch here to run process
  struct trapframe *tf;       // Trap frame for the
                              // current interrupt
};
```