Leonardo Vecchi Meirelles - 12011ECP002

Questão 1) Um processo é uma instância de um programa em execução. É a unidade básica de execução em um sistema operacional e consiste no código do programa, dados e contexto de execução.

Cada processo tem seu próprio espaço de memória, pilha, contador de programa e registradores de CPU. Ele executa independentemente de outros processos, embora possa interagir com eles por meio de chamadas de sistema ou meconismos de comunicação entre processos fornecidos pelo sistema operacional.

Leonardo Vecchi Meirelles - 12011ECP002

Questão 2) Processos podem pertencer a um dos seguintes estados:

- New: é o estado inicial de um processo quando ele é criado, mas ainda não foi admitido pelo sistema operacional para execução.
- Ready: o processo está esperando para ser executado pela CPU. Ele tem todos os recursos necessários para funcionar, mas a CPU está executando outro processo
- Running: o processo está sendo executado pela CPU. Apenas um processo pode estar neste estado por vez em um sistema de CPU única.
- Blocked: o processo está aguardando a ocorrência de algum evento antes de prosseguir. Por exemplo, pode estar esperando que os dados sejam lidos do disco.
- Terminated: o processo concluiu sua execução e foi finalizado pelo sistema operacional. Os recursos utilizados pelo processo são liberados pelo sistema operacional.

Leonardo Vecchi Meirelles - J2011ECP002

Questão 3) A ordem na qual os processos são executados em um sistema operacional pode ser influenciada por vários fatores, incluindo a política de escalonamento usada pelo sistema operacional, a prioridade dos processos e os recursos disponíveis no sistema.

A maioria dos sistemas operacionais modernos usa alguma forma de algoritmo de escalonamento de processo para determinar quais processos devem ser executados a seguir. Esses algoritmos geralmente levam em consideração fatores como prioridade do processo, utilização da CPU e tempo de espera para tomar decisões sobre quais processos executar.

Como o algoritmo de escalonamento leva em consideração uma variedade de fatores dinâmicos, a ordem na qual os processos são executados pode mudar de uma execução para outra, mesmo que o mesmo conjunto de processos esteja sendo executado. Isso significa que a sequência exata em que os processos são executados geralmente é imprevisível e pode depender de uma variedade de fatores que podem variar de momento a momento.

Leonardo Vecchi Meirelles - J2011ECP002

Questão 4) Para gerenciar processos, um SO geralmente precisa manter informações sobre cada processo que está em execução ou aguardando para ser executado. Essas informações podem incluir:

- Process ID (PID): um identificador exclusivo para cada processo.
- Parent Process ID (PPID): o PID do processo que criou esse processo.
- State: o estado atual do processo (new, ready, running, blocked ou terminated)
- Priority: o nível de prioridade atribuído ao processo.
- Program Counter (PC): o endereço na memória da próxima instrução a ser executada pelo processo.
- CPU registers: os valores armazenados nos registradores da CPU para o processo.
- Memory usage: a quantidade de memória atualmente sendo usada pelo processo.
- I/O status: o status atual de I/O do processo.
- File descriptors: os arquivos e outros recursos atualmente abertos ou sendo usados pelo processo.

Segue um exemplo de data structure em C:

```
struct Process {
    int pid;
    int ppid;
    char* state;
    int priority;
    void* program_counter;
    void* cpu_registers;
    size_t memory_usage;
    char* io_status;
    int* file_descriptors;
};
```

Leonardo Vecchi Meirelles - 12011ECP002

Questão 1) As três chamadas principais são:

• fork(): esta chamada cria um novo processo duplicando o processo de chamada. O novo processo é chamado de processo filho e é uma cópia exata do processo pai, incluindo sua imagem de memória e contexto de execução. Após a chamada fork(), os processos pai e filho continuam executando no mesmo ponto do programa, mas com IDs de processo diferentes. O valor de retorno da chamada fork() é usado para determinar se o processo atual é o processo pai ou filho.

• exec(): esta chamada substitui o processo atual por um novo processo. O novo processo é especificado por um arquivo executável, e a chamada exec() carrega o código do programa e os dados do novo processo na memória e começa a executá-lo. O processo chamador é totalmente substituído pelo novo processo e seus recursos são liberados. A chamada exec() é frequentemente usada em conjunto com a chamada fork() para criar novos processos com diferentes imagens de programa.

• wait(): esta chamada espera que um processo filho termine e retorna informações sobre o status de saída do processo filho. Se não houver nenhum processo filho em execução no momento, a chamada wait() será bloqueada até que um processo filho seja criado. A chamada wait() permite que o processo pai sincronize com seus processos filhos e determine quando eles concluirom a execução.

Leonardo Vecchi Meirelles - 12011ECP002

Questão 2) A chamada fork() retorna o ID do processo filho para o processo pai e retorna 0 para o processo filho. Isso permite que os processos pai e filho se diferenciem entre si.

O fluxo do programa pode ser dividido em dois caminhos: um para o processo pai e outro para o processo filho. O processo pai pode usar o ID do processo filho retornado por fork() para identificar e gerenciar o processo filho, enquanto o processo filho pode usar o valor de retorno 0 de fork() para se identificar como o processo filho.

Usando getpid() em cada processo, podemos imprimir o ID do processo pai e filho. A chamada wait() no processo pai garante que o processo filho conclua a execução antes que o processo pai seja encerrado.