UNIVERSIDADE ESTADUAL DE CAMPINAS
Instituto de Computação
MC504 Sistemas Operacionais
Prof. Isaías Bittencourt Felzmann

Exercícios de Revisão 1

Introdução

1) Cite quais são as principais funções de um Sistema Operacional.
2) Uma Chamada de Sistema é um dos pontos de entrada para que o Sistema Operacional atenda a uma requisição da aplicação de usuário. Explique o papel de uma Troca de Contexto para possibilitar isso.
3) Dê exemplos de atividades que podem ser executadas em modo de execução de usuário ou devem ser executadas em modo núcleo (modo kernel ou modo máquina). O que há de diferente entre os dois modos que determinadas atividades **podem** estar em um, enquanto outras **devem** estar em outro?
4) Um sistema tem uma CPU com apenas um core. Nesse sistema, queremos executar três processos, P_0, P_1 e P_2, com tempos de execução em modo usuário de 5 ms, 10 ms e 20 ms. Na média, cada processo executa uma chamada de sistema a cada 1 ms. O escalonador do Sistema Operacional atua a cada 10 µs. O tempo médio de uma troca de contexto é de 1 µs. O tempo de tratamento de uma chamada de sistema específica é desprezível. Quanto é o tempo relógio total para execução dos três processos nesse sistema? Mostre como você chegou até a resposta.

Processos e Threads

5) Um processo é uma abstração interna do Sistema Operacional que encapsula diversas informações sobre um programa em execução. Dê exemplos de informações que o Sistema Operacional guarda sobre cada processo em execução.
6) Quando um processo é dividido em várias threads, algumas das informações armazenadas sobre o processo são compartilhadas entre todas as threads, e outras são replicadas e armazenadas para cada uma das threads do processo. Dê exemplos de informações para os dois casos.
7) Um shell é uma aplicação de modo usuário utilizada para iniciar a execução, interativamente, de outros programas conforme solicitado pelo usuário. Para isso, o shell cria um novo processo e executa o novo programa dentro dele. Utilizando as chamadas de sistema e os padrões de chamada *Linux* estudados em aula, **esboce** um código em C para fazer a tarefa do shell de iniciar um novo programa. Você pode assumir que existe uma função de biblioteca `GetPgmInfo()` que devolve todas as informações necessárias sobre o programa solicitado pelo usuário já no formato necessário exigido pela chamada de sistema adequada.
8) Seria viável e aconselhável implementar a aplicação do shell, do exercício anterior, utilizando criação de novas threads ao invés de novos processos? Por quê?
9) Um processo com PID=2600 cria um novo processo utilizando o código a seguir. O processo criado recebeu, do Sistema Operacional, o PID=2603. Suponha que exista uma função `getpid()` que devolve o valor do PID do processo que a chamou. Dê duas possíveis saídas impressas no terminal para a execução do programa.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main(){
      pid_t pid, pid1;

      /* cria um processo-filho */
      pid = fork();
      if (pid < 0){
            /* um erro ocorreu */
            printf("Fork Failed\n");
            return 1;
      }
      else if (pid == 0) {
            /* processo-filho */
            pid1 = getpid();

            printf("child: pid = %d\n", pid);
            printf("child: pid1 = %d\n", pid1);
      }
      else {
            /* processo-pai */
            pid1 = getpid();

            printf("parent: pid = %d\n", pid);
            printf("parent: pid1 = %d\n", pid1);
            wait(NULL);
      }
      return 0;
}
```

10) Analisando o código abaixo, o que você pode afirmar sobre o número de vezes que a mensagem "Hello world!" será escrita na tela, de acordo com o valor de N?

```c
#include <stdio.h>
#include <unistd.h>

int main() {
      int i;

      for (i = 0; i < N; i++) {
            fork();
      }

      printf("Hello world!\n");

      return 0;
}
```

11) **Esboce** o código de uma aplicação multithread, usando threads Posix, para executar a multiplicação de duas matrizes quadradas `C[n][n] = A[n][n] x B[n][n]` com elementos inteiros. Assuma que o valor de `n` é uma constante em tempo de compilação ("`define`"), e você não precisa se preocupar com a alocação e inicialização das matrizes `A`, `B` ou `C`.
O seu código deve definir uma função `multiplica(...)` que recebe, como parâmetros, apenas três ponteiros `int*` (que são endereços dentro das matrizes `A`, `B` e `C`) e não retorna nada. A função deve se encarregar do resultado da multiplicação para um único elemento `C[i][j]`, ou seja, multiplica e acumula uma linha de `A` por uma coluna de `B` e armazena o resultado no elemento de `C`.

A rotina principal do seu programa deve iniciar a execução **concorrente** de n^2 Threads, uma para cada execução da função multiplica e preenchimento de um elemento `C[i][j]`. Lembre-se de fazer todos os ajustes necessários para passagem correta dos argumentos da função `multiplica`.

Escalonamento de Processos

12) Explique a diferença de escalonamento preemptivo e não-preemptivo.
13) Suponha que os processos da tabela a seguir cheguem para execução nos tempos indicados. Cada processo executará pelo tempo da sua duração de pico.

Processo	Tempo de chegada	Duração do pico
P1	0	8
P2	3	4
P3	10	1

(a) Desenhe um esquema do escalonamento desses processos considerando escalonamento FCFS (*first come, first served*). Qual é o tempo médio de espera para os três processos?
(b) Desenhe um esquema do escalonamento desses processos considerando escalonamento SRTF (*shortest remaining time first*, versão preemptiva de *shortest job first*). Qual é o tempo médio de espera para os três processos?
(c) Desenhe um esquema do escalonamento desses processos considerando escalonamento RR (*round robin*) com *quantum* de 2 unidades de tempo, em que todos os três processos têm o mesmo nível de prioridade.

Sincronização de Processos

14) Explique o conceito de condição de corrida e como ele se relaciona com o conceito de seção crítica.
15) Quais são as três características de uma solução do problema da seção crítica, e o que elas significam?
16) Um programa qualquer precisa gerar um arquivo de log relatando todas as operações que foram feitas. O nome e caminho desse arquivo de log está escrito diretamente no código da aplicação e é sempre o mesmo, não importando quantas vezes o usuário executa o programa. Para criar uma mensagem de log, o programa utiliza uma função `void log(char* msg)`. Suponha que exista uma segunda função `void writeToLogFile(char* msg)` que abre o arquivo de log em modo de escrita, escreve a mensagem `msg` e fecha o arquivo, apenas. Suponha, também, que o Sistema Operacional permite que vários processos separados mantenham um mesmo arquivo aberto ao mesmo tempo, sem nenhuma restrição. **Esboce** o código da função `log(...)` de modo a garantir que as mensagens sejam escritas de maneira legível para um usuário humano.
17) Em um sistema produtor-consumidor, um processo (ou thread) gera (produz) dados que serão utilizados (consumidos) por outro processo (ou thread) para continuar o processamento da aplicação. Esses dados são armazenados em uma região de memória compartilhada que pode armazenar uma estrutura de dados em fila, por exemplo. Você possui uma biblioteca que armazena uma estrutura de dados em fila em memória compartilhada que expõe as seguintes funções:

`void enfileira(item* x)`	Adiciona um elemento no fim da fila.

`item* desenfileira()`	Remove um elemento do início da fila. Retorna NULL se a fila estiver vazia.

Utilizando as funções da biblioteca acima, **esboce** trechos de código para os programas produtor e consumidor, de modo de minimizar o tempo de CPU em espera em ação (*busy wait*) na sua aplicação. Note que o sistema pode ter vários processos produtores e vários consumidores, todos eles executam os mesmos códigos, todos eles operam sobre a mesma fila compartilhada, a produção e o consumo de um dado levam tempos arbitrários e não previsíveis, e nenhum desses fatores deveria influenciar na sua implementação de uma solução.

Deadlock

18) Quais são as quatro condições necessárias para que um deadlock aconteça e o que elas significam? Dê um exemplo de situação que mostre que as quatro condições são necessárias, mas não suficientes – ou seja, dê um exemplo de uma situação que cumpra as quatro condições mas não seja um deadlock.
19) Um sistema tem três recursos compartilhados, R1, R2 e R3, cada um com uma instância, e executa três processos, P1, P2 e P3. A tabela a seguir mostra uma sequência de chamadas em que um processo solicita um determinado recurso, na ordem em que essas chamadas devem acontecer.

Chamada #	Processo	Recurso
1	P1	R2
2	P2	R3
3	P2	R2
4	P3	R1
5	P1	R1
6	P3	R3
7	P3	R2
8	P1	R3

(a) Assumindo que o sistema não implemente mecanismo para impedir deadlocks, desenhe o grafo de alocação de recursos para o sistema. O sistema entra em deadlock? Quais das chamadas são efetivamente executadas e atendidas?
(b) Assumindo que o sistema implemente mecanismo para impedir deadlocks e que, ao início, todos os processos declaram que podem utilizar todos os recursos, desenhe o grafo de alocação de recursos para o sistema. O sistema entra em deadlock? Quais das chamadas são efetivamente executadas e atendidas?