Semáforos

Complete o código seguinte para garantir que é sempre impresso o valor correto (3000000). Utilize os semáforos definidos na norma POSIX.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int cont = 0;



void *worker(void *arg) {
     int n, c;
     n = (int)arg; c = 0;


     for( i= 0; i < n; i++ )


          c ++;


     cont = cont + c;


}


int main(int argc, char *argv[]) {
    pthread_t p1, p2, p3;



    pthread_create(&p1, NULL, worker, (void*)1000000);
    pthread_create(&p2, NULL, worker, (void*)1000000);
    pthread_create(&p3, NULL, worker, (void*)1000000);
    pthread_join(p1, NULL); pthread_join(p2, NULL); pthread_join(p3, NULL);

    printf("%d\n", cont);
    return 0;
}
```

Complete o programa seguinte por forma a garantir que é sempre impresso o valor **24**. Deve utilizar apenas as operações sobre semáforos definidas na norma POSIX

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>



int y = 5;
int x;

void *worker1(void *arg) {



     x = y - 1;


}
void *worker2(void *arg) {



     y = y * y;
```

```
}

int main(int argc, char *argv[])
{
    pthread_t p1, p2;



    pthread_create(&p1, NULL, worker1, NULL);
    pthread_create(&p2, NULL, worker2, NULL);
    pthread_join(p1, NULL); pthread_join(p2, NULL);

    printf("%d\n", x);
    return 0;
}
```

Considere um mecanismo de sincronização entre processos caracterizada pelas operações seguintes:

- *CriarBarreira(Nome, N)* define um barreira de N processos
- *AguardarBarreira(Nome)* bloqueia o processo que invoca a operação até que N processos tenham invocado esta operação.

Usando semáforos, escreva o código para *AguardarBarreira*. Suponha que a operação *CriarBarreira* criou previamente todas as estruturas de dados necessárias.

Apresente o pseudo-código das acções efectuadas pelas seguintes operações sobre semáforos, segundo a definição de Dijkstra, em que 'sem' indica um semáforo. Deve descrever todas as estruturas de dados auxiliares ou outras operações básicas de que necessite. Assuma que a indivisibilidade do código que apresentar está garantida:
a) Operação Wait(sem)
b) Operacão Signal(sem)

Considere o problema dos leitores e escritores, processos concorrentes com acesso a uma base de dados, através das operações ler(R) e escrever(R) em que R designa um valor lido ou escrito. Admita que estas operações, que processam o acesso aos registos da base de dados, já estão implementadas, mas não controlam as interferências devidas aos acesso concorrentes dos múltiplos clientes leitores e escritores.
a) Baseando-se em semáforos, segundo a definição de Dijkstra, e em comunicação por memória partilhada, apresente o pseudo-código das acções seguintes:
pedir_ler()
pedir_escrever()
terminar_ler()
terminar_escrever()
que devem ser invocadas respectivamente, pelos leitores e escritores, antes e depois de poderem começar a ler ou a escrever. Note que as operações pedir_ler() e pedir_escrever() devem ser bloqueantes, caso o leitor ou o escritor não possa prosseguir por a base de dados estar correntemente ocupada por outros processos.
b) Explique quais as propriedades da solução que apresentou em a), do ponto de vista da justiça (fairness) no tratamento dos pedidos dos leitores e escritores.

Gestão de memória

Considere um sistema operativo (SO) que suporta múltiplos utilizadores e em que cada utilizador pode lançar mais do que um processo. Este SO gere a memória física com base em partições criadas dinamicamente com dimensão igual à da imagem do processo que vai ser criado. Explique porque é que este sistema de gestão de memória está sujeito ao problema da fragmentação externa.

Um CPU emite endereços virtuais com 24 bits. A unidade de gestão de memória (MMU) suporta paginação, tendo cada página 2048 (2^{11}) bytes. Diga de que forma é que feita a conversão dos endereços virtuais em endereços físicos. Indique, justificando, quantas entradas tem a tabela de páginas de um processo.

Considere uma unidade de transformação de endereços baseada em páginas e em que o hardware mantém para cada página P o bit de referência R. R[P] está inicialmente a 0; sempre que é feita uma referência à página P, R[P] é colocado a 1. Há uma instrução máquina privilegiada que coloca R[P] a 0. Admita que sobre este hardware, o sistema operativo implementa uma estratégia de substituição de páginas de 2ª hipótese (*2nd chance page replacement algorithm*). Este algoritmo é usado sempre que é preciso escolher uma página V para ser substituída. Explique porque é que a página V escolhida é uma página que não é referenciada há muito tempo.

Explique como é que uma unidade de gestão de memória (MMU) com um registo base e um registo limite pode ser usada para efectuar recolocação de programas e para impedir que um processo tenha acesso a zonas da RAM que não fazem parte da sua imagem.

Um CPU emite endereços virtuais com 24 bits. A unidade de gestão de memória (MMU) suporta paginação, tendo cada página 8192 (2^{13}) bytes. Diga de que forma é que feita a conversão dos endereços virtuais em endereços físicos. Indique, justificando, quantas entradas tem a tabela de páginas de um processo.

Considere um sistema de gestão de memória em que a MMU (*Memory Management Unit*) suporta uma tabela de páginas, em que cada página tem um bit de validade V. Sobre este hardware, o sistema operativo suporta paginação a pedido. Descreva as acções efectuadas pelo SO quando recebe uma interrupção da MMU assinalando que o processo A tentou referenciar a sua página virtual P, tendo a entrada P da tabela de páginas do processa A o bit de validade a 0.

O que entende por um sistema de gestão de memória central que funciona com paginação a pedido? Explique como é que este sistema permite executar programas cuja dimensão é superior ao tamanho da memória física.

A gestão da memória física por partições apresenta vários inconvenientes. Um deles é a fragmentação da memória.

a) A fragmentação pode ser *interna* ou *externa*. Explique o que significa cada um destes termos ?
b) Num sistema com partições em número e dimensões fixas, que tipo de fragmentação ocorre? Justifique.
c) E num sistema com partições variáveis?

No LINUX a gestão de memória utiliza uma MMU baseada em páginas. Quando é feito um *fork()* como é preenchida a tabela de páginas do novo processo?

Explique os inconvenientes dos sistemas de gestão de memória física baseados em partições contíguas. Explique de que forma é que os sistema de gestão de memória física baseados em páginas contribuem para resolver esses problemas.

Considere um sistema de gestão de memória baseado em páginas, suportado em paginação a pedido.
a) É possível executar um programa cujo imagem tenha uma dimensão superior à da memória física? Se sim, em que condições?
b) Neste ambiente, explique, em detalhe, o que ocorre quando o processo corrente referencia uma página que não está em memória.

Suponha um S.O. que executa numa máquina que tem uma MMU baseada em páginas. A MMU contém um "Translation Lookaside Buffer" (TLB). Explique qual a utilidade do TLB. Diga o que acontece ao TLB sempre que se comuta de processo.

Num dado S.O. a gestão de memória utiliza uma MMU baseada em páginas e é utilizada paginação a pedido. Explique em que consiste a situação de "trashing". Diga também de que forma de pode detectar que um dado sistema entrou em "trashing".

Sistema de ficheiros e dispositivos de entrada / saída

No sistema operativo Linux, um processo lançado pelo utilizador U executa a chamada ao sistema
int f = open("/home/bruno/x.c", O_RDONLY);
Explique como é que o sistema operativo verifica que o ficheiro indicado existe.

Considere um disco que depois de formatado tem 65536 (2^{16}) blocos e em que cada bloco tem 4096 (2^{12}) bytes; os endereços dos blocos têm 16 bits (2 bytes). Suponha que neste disco foi colocado um sistema de ficheiros que usa uma estratégia de atribuição de espaço em disco indexada, em que os blocos atribuídos a um ficheiro são definidos por 8 endereços de blocos. Os endereços de 0 a 6 são endereços directos e o endereço 7 é o endereço de um bloco que contém endereços de blocos. Diga, justificando, qual é o tamanho máximo, usando esta estratégia de atribuição, que um ficheiro pode ter?

Considere a seguinte sequência de chamadas ao sistema

```
(1) int f = open( "/x.log", O_WRONLY | O_CREAT);
(2) write( f, '123456', 6);
(3) close(f);
```

Para cada uma das chamadas ao sistema explique que consultas e alterações são feitas na área de meta-dados do disco que suporta o sistema de ficheiros em que está o ficheiro *x.log*. Suponha que o ficheiro não existia antes da chamada ao sistema (1) e que o processo que faz as chamadas tem permissões para escrever na directoria raiz do sistema.

Considere um disco que depois de formatado tem 2^{24} blocos e em que cada bloco tem 4096 (2^{12}) bytes; os endereços dos blocos têm 32 bits (4 bytes). Suponha que neste disco foi colocado um sistema de ficheiros que usa uma estratégia de atribuição de espaço em disco indexada, em que os blocos atribuídos a um ficheiro são definidos por 8 endereços de blocos. Os endereços 0 a 5 são endereços directos e os endereço 6 e 7 são os endereços de dois blocos que contêm endereços de blocos. Diga, justificando, qual é o tamanho máximo, usando esta estratégia de atribuição, que um ficheiro pode ter?

Considere o sistema de ficheiros do MS-DOS que é baseado na FAT (File Allocation Table). Qual dos métodos de atribuição de espaço em disco aos ficheiros estudados é usado? Justifique. Indique, justificando, qual é a máxima dimensão de um disco lógico suportado por este método.

Considere o sistema de ficheiros EXTENDED 2 do Linux. Descreva os vários passos efectuados quando se apaga um ficheiro, dado o seu nome.

Indique os passos realizados quando se efectua uma chamada ao sistema que realiza uma operação de saída de dados síncrona sobre um periférico tipo bloco. Admita que o periférico assinala a conclusão da operação através de uma interrupção.

Considere o sistema de ficheiros do UNIX. Explique o que se passa, em termos das estruturas de dados que descrevem o estado do sistema de ficheiros quando se dá o comando
mv /d1/f1 /d1/f2 .
Suponha que antes de dar o comando a directoria *d1* e o ficheiro *f1* existem e o ficheiro *f2* não existe

---

Considere um S.O. da família UNIX. Explique para que servem as chamadas ao sistema *open()* e *close()*. Concretize para o caso de um ficheiro normal e de um ficheiro especial (periférico). Note que não se pretende uma descrição de como é que estas chamadas são implementadas.

Considere o sistema de ficheiros EXT2 do LINUX. Descreva os vários passos efectuados quando, no interpretador de comandos, se escreve **mv nome_lógico_1 nome_lógico_2.**

Compare as formas de atribuição de espaço em disco a ficheiros utilizadas pelo sistema de ficheiros FAT (Windows) e pelo UNIX. Descreva as principais estruturas utilizadas por cada um deles. Indique as vantagens e os inconvenientes dos dois métodos.