**Fundamentos de Sistemas de Operação – Teste T2 16/12/2009**

Sem consulta. **Duração total: 2h30 horas**

**Na 1a folha, deve indicar:** o nome, o número de aluno e o total de folhas que entregar.

**Em cada uma das restantes folhas deve indicar:** o número de aluno e o número de ordem da folha**.**

## Faça cada Questão numa Folha separada

**Questão 1.** Explique as diferenças, as vantagens e os inconvenientes dos seguintes modelos de comunicação disponíveis no sistema Unix. Nessa comparação, oonsidere as duas dimensões: (i) o modelo oferecido ao programador; (ii) os mecanismos que suportam a implementação.

**a)** comunicação por memória partilhada entre processos (processos criados por *fork*)
**b)** comunicação por filas de mensagens (*message queues* Unix)

**Questão 2.** Explique as diferenças entre os conceitos de Processo (tal como criado por *fork* no Unix) e de *Thread* (tal como criado por *pthread_create*). Para cada conceito, Processo e *Thread*, descreva em que consiste o contexto que define o respectivo ambiente de execução.

**Questão 3** Considere, num ambiente de multiprogramação Unix, um conjunto de N+1 processos concorrentes, cujos identifcadores (*process id*) são denotados por P0, P1, P2, ... PN. Admita que todos o processos já se encontram criados.

**Processo P0 - mestre:**

este processo está continuamente a gerar tarefas e a distribuí-las pelos N processos (trabalhadores) P1, ... PN, para estes as executarem;

para que o processo P0 possa distribuir as tarefas pelos trabalhadores, considera-se uma fila (F) de tarefas, a qual é globalmente acessível a todos os processos, de forma concorrente e tem uma capacidade limitada a um certo número de tarefas, dependente da implementação;

cada tarefa posta na fila é descrita por um descritor. Não precisa de se preocupar com a estrutura interna do descritor que descreve cada tarefa, excepto que cada descritor tem um tamanho fixo de *bytes*, definido por uma constante chamada SIZE.;

O processo P0 coloca cada tarefa na fila F, através da seguinte operação:

***colocar (tarefa *T);*** – O argumento T aponta o descritor da tarefa a colocar. No caso de a fila F estar cheia, o processo P0 tem de aguardar, bloqueado, até poder completar esta operação. Esta operação deve implementar toda a sincronização necessária para que o acesso à fila F seja correcto e coerente.

**Processos P1, ... PN – trabalhadores:**

cada uma das tarefas deve ser executada por um qualquer (mas apenas por um) dos processos P1,..., PN.

cada um dos processos P1, ... PN efectua um ciclo de dois passos, que repete indefinidamente,:

1- aguardar até obter uma tarefa da fila F
2- executar a tarefa obtida (esta acção assume-se já implementada e não nos interessa aqui);

Cada processo P1,,.., PN obtém cada tarefa da fila F, através da seguinte operação:

***obter (tarefa *T);*** – o processo aguarda, bloqueado, até que possa obter (e remover) uma tarefa da fila. O argumento T aponta o descritor da tarefa obtida e, então, o processo prossegue a execução do seu ciclo, conforme foi acima explicado. Esta operação deve implementar todas a sincronização necessária para que o acesso à fila F seja correcto e coerente.

Para cada uma das alíneas seguintes, pede-se que descreva em C (pseudo-código), a implementação da fila global F e das funções *colocar () e obter()*.

**a)** Nesta alínea, para representar e controlar o acesso à fila global F, apenas pode utilizar filas de mensagens (*message queues Unix System V*). Admite-se que as filas necessárias já se encontram criadas.

**Apresente em C (pseudo-código), a implementação das duas funções *colocar* e *obter*.**

Deve usar explicitamente as funções definidas pelas chamadas ao Unix para operar sobre *message queues*.

Indique todas as declarações de variáveis e funções auxiliares de que necessite. Nesta alínea, não pode utilizar memória partilhada nem semáforos.

**b)** Nesta alínea, para representar e controlar o acesso à fila global F, apenas pode utilizar comunicação por memória partilhada entre processos (shared memory Unix System V) e sincronização por semáforos, segundo as operações definidas por Dijkstra (P e V). Admite-se que as regiões de memória partilhada já se encontram criadas (assim, não precisa de usar as chamadas ao sistema shmget/shmatt, basta declarar as estruturas necessárias em pseudo-código).

> Admite-se que a fila F, representada em memória partilhada, tem uma capacidade máxima limitada a N descritores de tarefas.
>
> Admite-se que as operações auxiliares para de acesso à estrutura de dados que representa a fila F em memória, já estão implementadas (mas note que estas duas funções não tratam da sincronização dos acessos concorrentes):
>
> > *inserir(tarefa *T)* - insere um elemento - só pode ser invocada se a fila não estiver cheia,
> >
> > *remover(tarefa *T)* - remover um elemento - só pode ser invocada se a fila não estiver vazia,

**Apresente em C (pseudo-código), a implementação das duas funções *colocar* e *obter***

Indique todas as declarações de variáveis que assume partilhadas em memória, os semáforos necessários e os seus valores iniciais.

**c)** Nas mesmas condições da alínea **3-b)**, pretende-se agora modificar a operação ***obter()*** de modo a impôr um *Time-out*, tal que, no caso desta operação bloquear o processo invocador, este processo não deva esperar mais do que o tempo especificado no argumento *Time_out* (que indica um intervalo de tempo a esperar, em segundos):

> ***int obter(tarefa * T, int Time_out)***
>
> Esta função deve retornar um inteiro igual a 0, no caso de de o processo ser desbloqueado por ter conseguido obter uma tarefa antes do Time-out, ou então deve retornar o valor -1 se ainda estiver bloqueado quando for atingido o fim do período de *Time-out*
>
> Nesta alínea **(3-c)** deve utilizar o mecanismo de sinais (*signals)* definido no sistema Unix. Assuma, para este efeito, que as operações P() têm um comportamento face aos sinais igual ao comportamento das chamadas bloqueantes ao SO Unix.

**Com base no mecanismo de sinais (*signals*) definido no sistema Unix, apresente em C (pseudo-código), a implementação da função *obter()*, definida nesta alínea 3-c) e de outras funções auxiliares necessárias.**

**Questão 4.** Considerando o conceito de monitor, tal como disponivel numa linguagem de programação concorrente, **apresente em pseudo-código um monitor que implemente as operações P() e V() sobre semáforos,** tal como definidas por Dijkstra.