Fundamentos de Sistemas de Operação - Exame 11/01/2010

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

Questão 1. Diga quais as diferenças entre os pares de conceitos indicados em cada uma das seguintes alíneas:

a) "***file descriptor***" comparado com´´***inode***´´ no sistema Unix.
b) "***ficheiro normal***" comparado com ´´***directoria***`` no sistema Unix.
c) "***processo***" no sistema Unix, comparado com ´´***thread***´.
d) ´´**semáforo**´´ tal como definido por Dijkstra comparado com ´´**mutex**´´ como definido na interface *Pthreads(POSIX)*

Questão 2. Explique, detalhadamente, as acções das seguintes chamadas ao sistema Unix, explique o seu efeito sobre as estruturas de dados internas do núcleo do sistema Unix que ache mais relevantes e indique o significado dos seus argumentos:

a) ***open***
b) ***link***
c) ***signal***
e) ***fork***

Questão 3. Considere que, no sistema Unix, um processo foi criando processos filhos, por diversas invocações da chamada ao SO ***fork()***. Admita que, no momento da terminação do processo pai, é sempre invocada uma função ***new_exit()*** que desencadeia a seguinte sequência de acções:

(1) invoca a função ***terminating()*** - indica aos filhos a terminação do pai, baseando-se em comunicação por ***pipes***.

(2) invocação da chamada ao SO ***exit()***

Usando chamadas ao sistema Unix, **apresente**, em C, o pseudo-código das seguintes funções:

- ***void terminating()***

- int wait_parent() - bloqueia o processo invocador até que o seu processo pai assinale a terminação da sua execução, devolvendo, como valor de retorno, o identificador do processo pai.

Explique todas as hipóteses e estruturas de dados que assumir na sua resolução.

Questão 4. Considere, no SO Unix, um conjunto de processos concorrentes, com dois processos emissores de mensagens (de identificadores P1 e P2) e dois processos receptores de mensagens (de identificadores P3 e P4). Pretende-se implementar uma forma de comunicação unidireccional tal que qualquer dos processos P1 ou P2 possa enviar mensagens a um qualquer dos processos P3 ou P4. Cada mensagem é descrita por um descritor (*Mensagem*). Não precisa de se preocupar com a estrutura interna do descritor que descreve cada mensagem, excepto que cada descritor tem um tamanho fixo de *bytes*, definido por uma constante chamada MSIZE. A comunicação é realizada pelas seguintes funções:

enviar_mensagem(Mensagem *Msg, int P) - o processo invocador (P1 ou P2) envia a mensagem *Msg* a um processo identificado pelo argumento P. Os únicos valores possíveis do argumento P são P3 ou P4. O argumento *Msg* aponta um descritor correspondente ao conteúdo da mensagem a enviar. Esta função apenas pode ser invocada pelo processo P1 ou pelo processo P2. Esta operação tem um comportamento assíncrono.

Cada um dos processos P3 ou P4 pode receber mensagens através da invocação da função:

receber_mensagem (Mensagem *Msg, int P) - em que o argumento *Msg* aponta o descritor correspondente ao conteúdo da mensagem recebida e o argumento P indica o identificador do processo do qual se deve obter uma mensagem. Os únicos valores possíveis do argumento P são P1 ou P4. Esta operação é bloqueante.

A implementação utiliza, para o encaminhamento de mensagens entre P1/P2 e P3/P4, dois processos servidores que actuam como intermediários PA e PB, conforme se indica na figura seguinte. Admita que todos os processos P1, P2, P3, P4, PA, PB já se encontram criados.

Conforme se indica na figura, os processos P1, P2, P3 e P4 não podem comunicar directamente entre si:

os processos P1 e P2 só podem comunicar, exclusivamente, com PA;

os processos P3 e P4 só podem comunicar, exclusivamente, com PB;

os processos PA e PB podem comunicar entre si, para suportarem o encaminhamento de mensagens; mas PA não pode comunicar com P3 ou P4 e PB não pode comunicar com P1 ou P2..

a) Nesta alínea, pretende-se implementar as operações acima indicadas, nas seguintes condições:

-- para todas as comunicações entre processos, apenas pode utilizar filas de mensagens - *message queues Unix System V* e admite-se que as filas necessárias já se encontram criadas. Nesta alínea, não pode utilizar memória partilhada, nem semáforos, nem *pipes*.

-- **apresente** o pseudo-código, em C, das duas funções ***enviar_mensagem*** e ***receber_mensagem***, bem como todas as declarações de variáveis relevantes e todas as filas de que necessite,

-- **apresente** o pseudo-código, em C, dos dois processos servidores **PA** e **PB**.

b) Nesta alínea, preende-se implementar as operações acima indicadas, nas seguintes condições:

-- para todas as comunicações entre os processos, apenas pode utilizar memória partilhada (*shared memory Unix System V*) e sincronização por semáforos, segundo a definição de Dijktstra, Admite-se que as regiões de memória partilhada necessárias já se encontram criadas e os processos já se ligaram a elas. Admite-se que os semáforos necessários já se encontram criados. Nesta alínea, não pode utilizar *message queues Unix System V*, nem *pipes*.

-- **apresente** o pseudo-código, em C, das duas funções ***enviar_mensagem*** e ***receber_mensagem***, bem como todas as declarações de variáveis relevantes, incluindo as variáveis e as filas partilhadas em memória e os semáforos necessários, indicando os seus valores iniciais. Admite-se que cada fila, representada em memória partilhada, tem uma capacidade máxima limitada a N mensagens.

Admite-se que as operações auxiliares para acesso às estruturas de dados que representam cada fila em memória, já estão implementadas (mas note que estas duas funções não tratam da sincronização dos acessos concorrentes):

*inserir(Elemento *E)* - insere um elemento - só pode ser invocada se a fila não estiver cheia,

*remover(Elemento *E)* - remover um elemento - só pode ser invocada se a fila não estiver vazia,

-- **apresente** o pseudo-código, em C, dos dois processos servidores **PA** e **PB**.