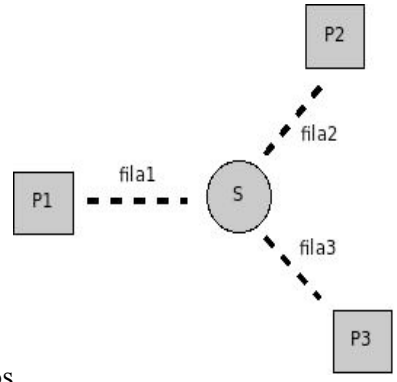
Fundamentos de Sistemas de Operação - Teste 17/12/2008
Sem consulta. **Duração total: 2h00 horas**

Questão 1. Considere um sistema de conversação (*chat*) que permite a troca de mensagens entre 3 processos (P1, P2,P3) que se assume que foram previamente criados. As mensagens (que na realidade são *strings*) são dirigidas a um servidor centralizado (S), também previamente criado, que tem como tarefa fazer a disseminação. Esta disseminação depende do tipo da mensagem: mensagens públicas devem ser entregues a todos os processos, excepto o que enviou; mensagens privadas devem entregues apenas ao destinatário. Cada processo `Pi` (i:1,2,3) **tem apenas acesso** às estruturas que representam a fila `filai` que o liga ao servidor. Nas alíneas a), b) e c), as filas são implementadas por filas de mensagens Unix e nas alíneas d) e e) são representadas por regiões em memória partilhada.



a) Pretende-se que os processos clientes P1, P2 e P3 tenham à sua disposição as seguintes operações:

- **`void enviaMensagemPrivada(char *mensagem, int proc)`** – envia uma mensagem privada para *proc*, onde *proc* é um número de 1 a 3 que identifica univocamente um processo. A função não é bloqueante, isto é, não espera que a mensagem seja entregue. No entanto, deve garantir que a mensagem é colocada na fila relevante.
- **`char* recebeMensagem()`** - recebe uma mensagem, com semântica bloqueante. Devolve um apontador para a mensagem.

Com base em filas de mensagens Unix (*message queues*) e nas respectivas chamadas ao SO Unix, implemente aquelas duas funções do cliente (nesta alínea não se pede o código do servidor). Apenas se pode usar uma fila para a ligação entre cada processo e o servidor. **Note:** cada processo `Pi` (i:1,2,3) **tem apenas acesso** à sua fila `filai` que o liga ao servidor. Assuma que essa fila já está criada e que o seu identificador se encontra na variável global `minha_fila`. Nesta alínea, não pode usar semáforos nem memória partilhada nem *pipes*.

b) Implemente a função **`void enviaMensagemPublica(char *mensagem)`** que envia uma mensagem pública para todos os processos excepto o próprio, também de forma não bloqueante (nesta alínea não se pede o código do servidor).

c) Apresente o código das acções do servidor S que processa os dois tipos de mensagens, assumindo que todas as inicializações estão feitas.

d) Em vez de filas de mensagens Unix, considere agora que a comunicação entre os clientes e o servidor assenta em regiões de memória partilhada, que se admitem previamente criadas e nas quais estão representadas as seguintes filas, um par de filas dedicado à comunicação entre cada processo e o servidor:

- `Fila filaA_i` fila em memória partilhada para as mensagens a receber pelo processo cliente Pi.

- `Fila filaB_i` fila em memória partilhada para as mensagens enviadas pelo processo cliente Pi.

Cada processo (`P1,P2,P3`) **tem apenas acesso** ao par de filas `filaA_i/filaB_i` que partilha com o servidor. Admita que cada fila apenas tem espaço limitado para conter um máximo de N mensagens, sendo N uma constante, e que todas as mensagens têm um tamanho fixo. Existem as seguintes funções disponíveis:

- `void P(int sem)` e `void V(int sem),` semáforos com a semântica definida por Dijkstra.
- `int colocaNaFila(Fila f, void* m)` – coloca uma mensagem `m` na fila `f`. Retorna 0 caso a mensagem seja colocada com sucesso ou -1 em caso contrário. Esta operação não controla os acessos concorrentes.
- `void* retiraDaFila(Fila f)` – Retira uma mensagem da fila `f`. Retorna NULL caso a fila esteja vazia. Esta operação não controla os acessos concorrentes.

Indique os semáforos de que necessita para sincronizar a comunicação entre um cliente e o servidor. Para cada semáforo indique o seu propósito e o seu valor inicial.

e) Nas mesmas condições da alínea d), implemente as funções `enviaMensagemPrivada e recebeMensagem, definidas na alínea a),` recorrendo às filas em memória partilhada dadas e a semáforos. Lembre-se que o envio é não bloqueante e que a recepção é bloqueante. Nesta alínea, não pode usar filas de mensagens Unix nem *pipes*.

Questão 2. Considere, no Unix, que um processo inicial P0 criou a seguinte configuração de dois processos filhos, P1 e P2, interligados através de um pipe e, depois disso, o processo P0 terminou.

Ficheiro F1 > P1 ----> Pipe pp1 ---> P2 > Ficheiro F2

em que se têm as seguintes ligações:
canal standard out de P1 mantém-se ligado ao lado de escrita de Pipe pp1
canal standard in de P1 mantém-se aberto para ler de F1
canal standard in de P2 ligado ao lado de leitura de Pipe pp1
canal standard out de P2 aberto para escrever em F2

Admita que os processos **P1** e **P2**, após estabelecidas as ligações indicadas, executam os programas contidos respectivamente nos ficheiros executáveis "**/tmp/fP1**" e "**/tmp/fP2**".

Pretende-se garantir que, em caso de o processo P2 terminar inesperadamente, através de acções equivalentes à chamada ao SO exit(), o processo P1 seja capaz de reagir, reconfigurando o sistema para o seguinte esquema:

Ficheiro F1 > P1 ----> Pipe pp2 ---> P3 > Ficheiro F2

em que P3 é um novo processo (que executa o ficheiro "**/tmp/fP2**") e pp2 é um novo pipe, tal que:
canal standard out de P1 ligado ao lado de escrita de Pipe pp2
canal standard in de P3 ligado ao lado de leitura de Pipe pp2
canal standard in de P1 aberto para ler de F1
canal standard out de P3 aberto para escrever em F2, a partir do ponto em que P2 estava a escrever em F2, quando terminou.

Usando chamadas ao sistema Unix, **apresente apenas o código, em C, que P1 deve executar** para garantir a reconfiguração do sistema nas condições apresentadas.