Das Questoes-SO-Autoavaliacao-II.pdf na Documentacao de apoio-> Testes

José C. Cunha – FSO – 4/12/2012

Indicam-se sugestoes de resolucao. Por favor note que sao apenas esqueletos de resolucao onde faltam por vezes as devidas declaracoes de variaveis e args das funcoes. Pretende-se apenas apontar caminhos de solucoes a desenvolver. Não fiz re-verificacao pelo que pode haver algumas gralhas das quais peço desculpa. Não se esqueça de que o objectivo das questões é o de treinar, por isso tente primeiro antes de ver as soluções. Em caso de dúvidas: jcc@fct.unl.pt

Questão 1. ----------

a)

```
int p1, p2, p3;
int fd1[2], fd2[2], fd3[2];

pipe(fd1); pipe(fd2); pipe(fd3);
p1 = fork();
if (p1==0) {   dup2(fd1[1], 1); dup2(fd3[0], 0); execv(P1exe, args);}
p2 = fork();
if (p2==0) {   dup2(fd2[1], 1); dup2(fd1[0], 0); execv(P2exe, args);}
p3 = fork();
if (p3==0) {   dup2(fd3[1], 1); dup2(fd2[0], 0); execv(P3exe, args);}
```

b)

```
int p1, p2, p3;
int fd1[2], fd2[2], fd3[2];
int ppc[2]; char c = ´x´; char buf[];

pipe(fd1); pipe(fd2); pipe(fd3);
pipe(ppc);
p1 = fork();
if (p1==0) {   dup2(fd1[1], 1); dup2(fd3[0], 0); read(ppc[0],buf, 1); execv(P1exe, args);}
p2 = fork();
if (p2==0) {   dup2(fd2[1], 1); dup2(fd1[0], 0); read(ppc[0],buf, 1); execv(P2exe, args);}
p3 = fork();
if (p3==0) {   dup2(fd3[1], 1); dup2(fd2[0], 0); read(ppc[0],buf, 1); execv(P3exe, args);}
write(ppc[1], &c, 1); write(ppc[1], &c, 1); write(ppc[1], &c, 1);
```

c)

```
int p1, p2, p3;
int fd1[2], fd2[2], fd3[2];
int status1, status2, status3;
pipe(fd1); pipe(fd2); pipe(fd3);
```

```
p1 = fork();
if (p1==0) {   dup2(fd1[1], 1); dup2(fd3[0], 0); execv(P1exe, args);}
p2 = fork();
if (p2==0) {   dup2(fd2[1], 1); dup2(fd1[0], 0); execv(P2exe, args);}
p3 = fork();
if (p3==0) {   dup2(fd3[1], 1); dup2(fd2[0], 0); execv(P3exe, args);}

wait(&status1); wait(&status2); wait(&status3);
```

d)

Sem considerer como indicar a mudanca pedida (recurso a sinais, ainda nao estudados), a reconfiguracao poder-se-iam fazer com outros 3 pipes adicionais que teriam de ser criados tambem antes dos forks() e que se reconfiguravam num programa semelhante ao acima, mas em sentidos contrarios.

Questao 2. ----------

a)

Por cada fila necessita de:

Semaforo Ex = 1

Semaforo Sc = 0

Semaforo Sv = N

b)

```
void enviaMensagemPrivada(char *mensagem, int proc)
 {// seja me = identificador do processo corrente
P(Sv-filaB-me)
P(Ex-filaB-me)
colocaNaFila(filaB-me, mensagem);
V(Ex-filaB-me)
V(Sc-filaB-me)
}


char* recebeMensagem()
{// seja me = identificador do processo corrente
P(Sc-filaA-me)
P(Ex-filaA-me)
mensagem = retiraDaFila(filaB-me);
V(Ex-filaA-me)
V(Sv-filaA-me)
}
```

Questao 3.----------

... a configuracao **Ficheiro F1 > P1 ----> Pipe pp2 ---> P3 > Ficheiro F2**

Se iniciada pelo processo P1, consegue-se assim: (omitindo declaracoes, identicas à Questao 1)

.... no processo P1...
pipe(pp2);
p3 = fork();
if (p3==0) { dup2(pp2[0],0); close(1); open(F2, O_WRONLY), execv(´/tmp/fP2´, args);}
// em P1
close(0); open(F1, O_RDONLY); dup2(pp2[1], 1);

Questao 4. ----------
Veja nos acetatos e apontamentos.

Questao 5. ----------
a)
ADMITA as seguintes operacoes sobre FILAS de MENSAGENS estao implementadas pelo SO pelo que as pode utilizar na solucao:

criarFila(nome-fila); // cria uma fila global de mensagens com o nome dado

sendM(char * Msg, char *Fila, int type) // envia Msg para Fila, com type dado – nao bloqueante execpto se a fila estiver cheia: neste caso aguarda até haver espaco para enviar

char * receiveM(char *Fila, int type) // devolve apontador para mensagem recebida, com type indicado, bloqueando o processo enquanto aguarda. O argumento type pode ter as seguintes valores:
type = QUALQUER (nesse caso recebe uma mensagem de qualquer tipo)
type =x um valor inteiro (nesse caso aguarda até receber uma mensagem desse tipo)
type = tipo X ou tipo Y ou... (uma disjuncao de valores, nesse caso pode receber uma qualquer mensagem dos tipos indicados)- A mensagem recebida tem tipo indicado em M.tipo

Assuma que a mensagem recebida M tem
os seguintes campos:
M.emissor – identifica o processo emissor
M.tipo – identifica o tipo inteiro da mensagem recebida

colocar (tarefa *T);
{
sendM(T, F, 1); // tarefas tem tipo = 1

}

obter (tarefa *T);
{
T = receiveM(F, 1); // recebe mensagens tipo = 1
}

NOTA: se pedissem que os trabalhadores pusessem os resultados depois de executarem, entao poderia ter as seguintes solucoes:
A) usar uma unica fila FR só para os resultados e ter cada trabalhador a fazer o seguinte, depois de executar a sua tarefa:
sendM(R, FR, 2); // admitindo que R representa o apontador para o descritor dos resultads e que as mensagens de resultados tem tipo=2.

B) usar a mesma fila F que ja era usada para o Mestre, e fazer com que o Mestre delegasse num filho a tarefa de coligir os resultados, assim:
-.. no Mestre ...

p = fork();

if (p ==0) { while(1) { ... receiveM(R,FR,2); ...}

---

5. b)

A fila F em memória partilhada, protegida por semaforos:
Ex = 1, Sc = 0, Sv = N, sendo N a capacidade da fila

colocar(T)
{
P(Sv)
P(Ex)
Inserir...
V(Ex)
V(Sc)
}

char recebeMensagem()*
P(Sc)
P(Ex)
Remover...
V(Ex)
V(Sv)
}

Questao 6. ----------
a)
Admita as mesmas operacoes basicas sobre filas de mensagens sendM e receiveM que na questao 5
Admita uma fila Fi por cada processo com i:1..N.

difundir_mensagem(char * Msg, int Wait)
{
if (wait == 1) { tipo = 1}
else {tipo = 2}

for (i=1;i<=N;i++) {
if (i != me) // admita me indica o identificador do processo corrente
sendM(M, Fi, tipo);
}
If (tipo == 1){
for (i=1;i<=N;i++) {
receiveM(M, F-me, 3); // recebe na sua fila, M mensagens de reply, tipo == 3

```
        }
}
```

```
receber_mensagem (char *Msg)
{
type = {1, 2}:
M = receive(F-me, type); // aguarda na sua fila, mensagens de tipo 1 ou 2
if (M.tipo == 1)
        sendM(Ok, Fila-M.emissor, 3); // envia reply Ok, para fila do emissor

}
```

b)

Cada processo tem uma fila-i em memoria partilhada, protegida pelos semaforos

Ex-i = 1

Sc-i = 0

Sv-i = N capacidade da fila

Cada processo tem um semáforo S-wait-me inicializado a 0 onde se bloqueia no caso de mensagens de tipo 1, para difusoes bloqueantes. Os outros processos (N-1) tem de fazer V sobre este semaforo.

```
difundir_mensagem(char * Msg, int Wait)
{// tipo 1 vai exigir confirmacao dos outros N-1 processos
if (wait == 1) { prepara mensagem M com M.text obtido de Msg, com M.tipo = 1}
else { prepara mensagem M com M.text obtido de Msg, com M.tipo = 2}

for (i=1;i<=N;i++) {
        if (i != me)  // admita me indica o identificador do processo corrente
                P(Sv-i); P(Ex-i);
                colocar(M, Fila-i);
                V(Ex-i); V(Sc-i);
        }

for (i=1;i<N;i++) {
        P(S-wait-me); // faz N-1 Ps sobre este semaforo
        }
}

receber_mensagem (char *Msg)
{
        P(Sv-i); P(Ex-i);
        retirar(M, Fila-me);
        V(Ex-i); V(Sc-i);
        if (M.tipo == 1)
                V(S-wait-M-emissor; //  faz V no semaforo S-wait do emissor

}
```

Questao 7.----------

a) veja apontamentos
basta fazer fork e no pai fazer exit e no filho fazer return
b) basta fazer fork e no pai fazer exit e no filho fazer exec ("/usr/bin/f", args);
Questao 8.---------- Identica a 1.

Questao 9.----------
O processo P4 pode receber mensagens de P2 ou P3 pela fila FM fazendo inicialmente
Msg = receiveM(FM,type); em que type ={2, 3} ou seja pode receber mensagens de 2 ou de 3.
Mas apos receber a primeira mensagem, deve ver de que tipo era:
Se Msg.tipo = 2, entao a proxima a receber deve ser do P3, pelo que fara a seguir
receiveM(FM,3)
Se Msg.tipo = 3, entao a proxima a receber deve ser do P2, pelo que fara a seguir
receiveM(FM,2)
e assim sucessivamente.

Questao 10.----------
Semaforos necessarios:
ExB = 1 exclusao mutua ao bloco
ScheiasB = 0 contador bloco: 0 – vazio, 1- preenchido
SvaziasB = 1 capacidade do bloco = 1

porBloco(buf)
{
P(SvaziasB) // consegue por um bloco no inicio, depois tem de aguardar se voltar aqui
P(ExB);
Inserir...
V(ExB);
for (i=1;i<N;i++) // avisa os leitores (N-1) de que já podem ir ler
V(ScheiasB);
} // note que normalmente só faria um V(ScheiasB) pois só colocou um bloco, no entanto pretende-se
// que os leitores que sao N-1 possam todos ler este bloco, pelo que vamos fazer um P(ScheiasB)
//no inicio de cada leitor, sendo assim precisamos de fazer N-1 Vs do lado do produtor
// ou seja apos pôr um bloco, o valor de ScheiasB vale N-1

lerBloco(buf)
{
P(ScheiasB); // só prossegue se o produtor já fez V(ScheiasB), ou seja colocou um bloco
chegada-barreira();
}
Em que a funcao chegada-barreira() implementa uma barreira de sincronizacao (veja os acetatos – exemplo 12 dos exemplos da programacao concorrente com semaforos), devida adaptada ao caso;

chegada-barreira() // assuma int count = 0; semaforo exmut = 1, semaforo block = 0

```
{
  P(exmut);
  count++; // indica que chegou
   if (count<N-1)   // nao é o ultimo leitor a ler o bloco
     {     lerbloco();  // lê
           V(exmut); // liberta para evitar deadlock
           P(block);}; // bloqueia-se
  else { lerbloco(); // este é o ultimo
           for (i=1; i<N-1; i++) V(bloq); count = 0;} // desbloqueia os outros
           V(SvaziasB); // avisa o produtor de que todos leram
           V(exmut);}
}
```

Questoes 11 e 12 ---- ver acetatos e apontamentos

Questao 13. ----------
Basta fazer:

close(0); open(filename_in, O_RDONLY);
close(1); open(filename_out; O-WRONLY);
….

Questao 14. ----------

No inicio o pai cria um pipe(pp) antes de fazer fork.

E o filho para fazer wait_parent() só tem de fazer read(pp[0], buf, 1); ficando aguardar que o pai
Ao chamar terminating() escreva ali um carácter qualquer fazendo write(pp[1], &c, 1)

Questao 15. ----------

O procS para ler das filas fM1 e fM2 deve criar dois processos filhos e delegar num a recepcao de mensagens de fM1 e no outro a recepcao de fM2. Assim cada um destes filhos pode enviar as suas mensagens para a fila fM3. O filho que lê de fM1 deve enviar mensagens com sendM(M, fM3, 1) isto e com tipo = 1 para indicar serem de P1. O filho que lê de fM2 deve enviar mensagens com sendM(M, fM3, 2) isto e com tipo = 2 para indicar serem de P2.
O processo P3 pode assim ler:
Qualquer mensagem de fM3, fazendo
        mensagem = receiveM(fM3, QUALQUER)
Uma mensagem de P1, fazendo
        mensagem = receiveM(fM3, 1) isto e type=1
Qualquer mensagem de P2, fazendo

mensagem = receiveM(fM3, 2) isto e type = 2

Questao 16.

Como na Questao 15, o procS para ler das filas fM1 e fM2 deve criar dois processos filhos e delegar num a recepcao de mensagens de fM1 e no outro a recepcao de fM2. E depois a solucao é identica aos exemplos anteriores que usam filas em memoria partilhada.

Questao 17.----------

Sugestao: use dois pipes criados inicialmente pelo pai antes de fazer fork
pipe(pp1) e pipe(pp2)
depois:
-- no programa de P1 a funcao aguardar-parceiro corresponde a:
escrever um caracter no pipe pp1
ler um caracter do pipe pp2
-- no programa de P2 a funcao aguardar-parceiro corresponde a:
escrever um caracter no pipe pp2
ler um caracter do pipe pp1

Questao 18. ----------

A solucao é identica às das questoes que implementam filas em memoria partilhada usando semaforos.