2º Teste FSO - 2012-13
Questão 2-a)

```
#define N       …
#define K1      1024

typedef struct {
        int slotsVazios, slotsCheios, mutex;
        char buffer[N][K1]
} FilaSHM;
typedef Fila *FilaSHM;

Fila Fila1-2, Fila2-3, Fila2-4;

// Inicializações dos semáforos: para cada FilaX-Y, slotsVazios= N, slotsCheios= 0, mutex= 1

void enviaBloco( char *bloco, Fila f) {
        P(f->slotsVazios);
        P(f->mutex);    colocaNaFila(bloco, f);  V(f->mutex);
        V(f->slotsCheios);
}

// Quem chama, faz free() depois de usar. Assume que retiraDaFila devolve um pointer para o
buffer, e ainda é preciso copiar o conteúdo
char * recebeBloco(Fila f) {
        char *ptr;
        P(f->slotsCheios);
        ptr=malloc(K1*sizeof(char));
        P(f->mutex);    memcpy(ptr, retiraDaFila(f));  V(f->mutex);
        V(f->slotsVazios);
        return ptr;
}
```

Teste T2 – FSO 2012-13 – Sugestoes de resolucao

Q1 a)

```
 . . .

int p, p1_2[2],  p2_3[2],  p2_4[2];

pipe(p1_2);

if ( (p=fork()) == 0 ) {     // P1
      close(0);
      open("F",O_RDONLY);
      dup2(p1_2[1], 1);
      close(p1_2[0]);
      close(p1_2[1]);
      Funcao_P1();
      exit(0);
}

pipe(p2_3);
pipe(p2_4);

if ( (p=fork()) == 0 ) {     // P2
  dup2(p1_2[0], 0);
  close(p1_2[0]);
  close(p1_2[1]);
      Funcao_P2();
      exit(0);
}

close(p1_2[0]);
close(p1_2[1]);

if ( (p=fork()) == 0 ) {     // P3
  close(p2_4[0]);
  close(p2_4[1]);

  dup2(p2_3[0], 0);
  close(p2_3[0]);
  close(p2_3[1]);
      Funcao_P3();
      exit(0);
}

if ( (p=fork()) == 0 ) {     // P4
  close(p2_3[0]);
  close(p2_3[1]);

  dup2(p2_4[0], 0);
  close(p2_4[0]);
  close(p2_4[1]);
      Funcao_P4();
```

```
      exit(0);
}

close(p2_3[0]);
close(p2_3[1]);
close(p2_4[0]);
close(p2_4[1]);

 . . .
```

Q1 b)

```
Funcao_P2() {
      char buf[1024];
      int n;

      while ( (n=read(0, buf, 1024)) > 0 ) {
            write( p2_3[1], buf, n );
            write( p2_4[1], buf, n );
      }
}
```

Q3 a)

Modelo de programacao: tem 4 diferencas.

a)
Pipes: locais a processos da mesma familia, por nao terem nomes globais (esta pergunta refere-se a pipes Unix, e nao demos os named pipes)
MemPartilhada; com ids globais e acessiveis a quaisquer processos com permissoes que facam shmget/attach

b)
Pipes: modelo de stream: fluxos de bytes contiguos preservando a ordem, sequencia de insercao no pipe
MemPartilhada: modelo de leitura e escrita, acesso directo, qualquer ordem

c)
Pipes: com leitura destrutiva, os bytes lidos são consumidos
MemPartilhada; leitura não destrutiva

d)
Pipes: operacoes read/write atómicas e com sincronizacao implicita garantida pelo SO; read sincrono ou bloqueante se pipe vazio e write bloqueante se pipe cheio; - portanto sao mais faceis de usar para o programador
MemPartilhada: leitura e escrita directas via apontadores, sem sncronizacao nem atomicidade garantidas pelo SO,
- mais dificil para o programador

Implementacao: tem 2 diferencas
a)
Pipes: buffers geridos em zonas de memoria do SO
MemPartilhadas: zonas de memoria fisica partilhada, via apontadores das tabelas de paginas

b)
Pipes: comunicacao entre dois processos envolve duas cópias: de emissor (write) para SO, e de SO para receptor (read)- por isso menos eficiente do que memoria partilhada
MemPartihada; comunicacao não envolve copia de bytes; o acesso é directo à memoria fisica - por isso mais eficiente do que pipes.

Exemplo1: pipes preferivel a mempartilhada: uma aplicacao de streaming--
Exemplo2: mempartilhada preferivel: uma aplicacao de bases de dados ou com tabelas ou com estruturas de dados que sao lidas e escritas

Q3 b)

Uma solucao dos slides
(mas é correcta qualquer solucao que garanta que se tem sempre:
Valor Inicial + Nº operacoes V efectuadas >= Nº operacoes P completadas)

Estruturas de dados auxilares:
um inteiro S nao negativo representa o valor do semaforo, inicializado pela operacao de criacao
uma fila para os identificadores dos processos bloqueados no semaforo, inicialmente vazia

b1)
```
P(S): (*operaçãoindivisível*)
if S>0 thenS = S-1
else
begin (*bloqueiaesteprocesso*)
salvaguardaestado/regsdamáquina;
estado(processo) = BLOQUEADO;
inserir(processo, filaf(S));
rotina-despacho-SO escolheoutroprocessodafilaPRONTOS, paraexecução
end
```

b2)
```
V(S): (*operaçãoindivisível*)
if (filaf(S) VAZIA) thenS = S+1
else
begin (*desbloqueia um processo*)
remover(filaf(S), Proc); -- remove Proc da fila
```

```
inserir(Proc, filaPRONTOS);
estado(Proc) = PRONTO
end
```

Sendo as ops P e V implementadas como chamadas ao SO, a sua indivisibilidade fica garantida com o SO a garantir que dois ou mais processos nao podem executar operacoes P ou V concorrentemente sobre um mesmo semaforo, conseguido pelo SO inibindo a comutacao entre processos durante a sua execucao, ou seja quando um processo está a executar o código de P ou de V sobre um semaforo S, mais nenhum pode iniciar a execucao desse código sobre o mesmo semaforo enquanto o primeiro processo nao se bloquear ou retornar.

Q3 c)

(1) substitui o mapa de memória corrente do processo, por um novo mapa de memoria correspondente ao novo programa, com novas zonas de codigo e dados, obtidas a partir do ficheiro executavel indicado como argumento de execve, e com nova zona de stack inicializada de novo. Quanlquer memória partilhada que estivesse mapeada nesse mapa, deixa de o estar como se tivesse sido feito "detach" e, logo, deixa de ser acessível por este processo;
(3) mantém a tabela de canais inalterada porque o processo continua o mesmo;
(4) o processo continua em execução, sendo o estado do CPU iniciado para o novo programa e novo mapa de memória;
(5) o identificador continua o mesmo porque o processo continua o mesmo;
.