Exame (N) de Fundamentos de Sistemas de Operação
Ano lectivo 2011/12, 1º semestre

Sem Consulta. Duração total: 2h30

Parte II

1. [7 valores] Considere a seguinte aplicação, formada por dois programas (i e ii). O programa (i) cria uma fila de mensagens e envia uma sucessão de 10 mensagens (10 termos inteiros 0, 1, ... 9) para a fila; o programa (ii) acede à fila, lê as mensagens e calcula o somatório dos termos recebidos. Note que os programas são executados individualmente (se quiser, em janelas distintas), isto é, um **não** faz `fork()` do outro.

(i)

```
#define CHAVEi ...
#define OPCAOi ...
#define SIZEi ...
#define BLOQi ...

int main() {
  int i, fi;

  struct {
    long mtype;
      ...
  } msgi;

  // criar fila de mensagens
  fi=msgget(CHAVEi, OPCAOi);

  msgi.mtype= ...;
  for (i= 0; i < 10; i++)
    msgi.(...)= i;
    msgsnd(fi, &msgi, SIZEi, BLOQi);
  }
  return 0;
}
```

(ii)

```
#define CHAVEii ...
#define OPCAOii ...
#define SIZEii ...
#define BLOQii ...

int main() {
  int i, fii, soma= 0; int tag= ...;

  struct {
    long mtype;
    ...
  } msgii;

  // usar fila de mensagens criada por (i)
  fii=msgget(CHAVEii, OPCAOii);

  for (i= 0; i < 10; i++) {
    msgrcv(fii, &msg, SIZEii, tag, BLOQii);
    soma= soma+msgii.(...);
  }
  printf("Soma: %d", soma);
  return 0;
}
```

a) Porque ordem têm de ser executados os programas (i) e (ii) para que se obtenha o resultado pretendido? Justifique.

b) Preencha os pontos (espaços com ...) de forma que os programas fiquem completos e funcionem.

c) Considere agora que imediatamente antes do fim de cada programa (i.e., antes dos `return 0`) queremos inserir dois novos ciclos `for`, idênticos aos anteriores – isto é o ciclo em (i) envia 10 elementos e o ciclo em (ii) recebe esses elementos e calcula o somatório dos seus elementos. Só que, agora, os elementos não são números inteiros, mas sim reais (`float`).

(i)

```
...

msgi.mtype= ...;
for (i= 0; i < 10; i++)
  msgi.(...)= i;
  msgsnd(fi,&msgi,SIZEi,BLOQi);
}

// novo ciclo p/ enviar float's

return 0;
}
```

(ii)

```
...

for (i= 0; i < 10; i++) {
  msgrcv(fii,&msg,SIZEii,tag,BLOQii);
  soma= soma+(msgii.(...))^2;
}
printf("Soma: %d", soma);

// receber floats, calcular somatório

printf("Soma: %f", somaf);
}
```

Altere os programas para que estes cumpram os objectivos enunciados. Decida se quer criar um novo tipo para as mensagens, ou se aproveita (e como) os tipos anteriores, `msgi` e `msgii`.

d) Altere o programa (i) de forma a que cada ciclo seja executado por uma thread independente; as duas threads deverão ser executadas concorrentemente, e o programa (i) só poderá terminar quando as duas threads tiverem terminado.

e) Atendendo à concorrência entre threads, as mensagens contendo números inteiros vão ficar "misturadas" na fila com as mensagens contendo números reais. Altere o programa (ii) para que este continue a calcular os dois somatórios (de inteiros e reais) correctamente, independentemente da ordem com que as mensagens foram depositadas na fila (se necessário, altere também o programa (i)).

2. [4 valores] Considere o sistema de ficheiros ext2 do Linux.

   a) De que forma é feita a contabilização dos blocos livres/ocupados?

   b) Descreva a estrutura de dados conhecida como i-node, indicando para que, e como, é usada.

   c) Descreva a constituição (isto é, a estrutura de dados, o tipo) e o objectivo de cada um dos campos que existem numa entrada de directoria de um sistema de ficheiros ext2.

3. [4 valores] Considere a execução de processos no Unix (ou num seu descendente, como o Linux).

   a) Defina sucintamente o conceito de processo.

   b) Desenhe o diagrama de estados de um processo, bem como as transições entre esses estados, caracterizando sucintamente cada estado – por exemplo, descrevendo que condições têm de se verificar para o processo estar nesse estado – e cada transição – para esta descrevendo, por exemplo, que condições têm de se verificar para que a transição se dê.

   c) Diga em que consiste a comutação de processos (process switching), e quais as informações fundamentais que têm de ser preservadas para salvaguardar o estado de um processo.