Fundamentos de Sistemas de Operação MIEI 2016/2017

Exame de Recurso - 19 de Janeiro de 2017– Duração 2h30m /1h 45m

Quem faz o exame deve resolver as questões: 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13

Quem faz o primeiro teste, faz as questões 1 a 7 (será feita a proporção para 20)

Quem faz o segundo teste, faz as questões 8 a 13 (será feita a proporção para 20)

Nome: Número:

Avisos: *Sem consulta; a interpretação do enunciado é da responsabilidade do estudante; se necessário explicite na resposta a sua interpretação. Ver na última página informação sobre chamadas ao sistema e algumas bibliotecas.*

Questão 1 – 2,0 valores

Um sistema operativo que suporta múltiplos processos (multi-programado) e múltiplos utilizadores (multi-utilizador) tem de garantir que erros que ocorram num processo não afectam o comportamento de outros processos. Que hardware e software é que são necessários para para garantir o seguinte:

a) Impedir que os dados de um processo sejam alterados por acção de outro processo.

O SO cria uma máquina virtual independente, com CPU, RAM e canais de entrada/saída, para cada processo. Para assegurar que a RAM atribuída a um processo só é acessível a este é preciso um MMU que assegure a deteção de acessos ilegais; esta MMU só pode ser programada pelo SO.

b) Impedir que os ficheiros de um utilizador sejam lidos ou escritos por um outro utilizador.

Os ficheiros guardados no disco só são acessíveis através de chamadas ao sistema. O SO pode verificar se o acesso é legal baseado nos bits de proteção que o cada associ ao ficheiro. Para impedir o acesso direto ao disco é preciso que o CPU tenh dois modos e que as instruções de E/S sejam privilegiados.

Questão 2 – 2,0 valores

Muitas arquitecturas hardware suportam a instrução máquina *interrupt return* (iret). Esta instrução muda o modo do CPU de modo sistema para modo utilizador

a) Em que parte do sistema operativo é que esta instrução é utilizada ?

A instrução iret é usada no final das rotinas de tratamento de interrupções, quer resultantes de hardware quer de interrupções por software; estas últimas são usadas para suportar as chamadas ao sistema.

b) A instrução *iret* é uma instrução privilegiada? Justifique.

Não precisa de ser uma instrução privilegiada, uma vez que leva o CPU para modo utilizador.

Questão 3 – 1,5 valores

Como é sabido, quando um utilizador escreve na linha de comando do *shell* a sequência

`nome_de_ficheiro_executável > nome_de_ficheiro`

o shell

- Cria um processo *p1* para executar `nome_de_ficheiro_executável`
- Redirige o canal *stdout* de *p1* para o ficheiro *nome_de_ficheiro*

Suponha que o fragmento de código seguinte foi retirado do código da *shell*. Complete os troços em falta

```
char *args[MAXARGS];
int p, status;
// suponha que foi chamada a função makeargv e que args contém apontadores para as
// três componentes da linha e que as 3 strings estão devidamente terminadas.
...
if ( strcmp( args[1], ">" ) == 0)
    p = fork();
    if (p > 0) { // sem teste de erro
            p = open(args[2], O_WRONLY);
            close(1);
            dup(p);
            args[1] = NULL;
            execvp(args[0], args);
    } else {

            wait(&status);

    }
}
```

Questão 4 – 2,0 valores

Complete o seguinte programa que pretende utilizar 4 *threads* para calcular quantos elementos de uma matriz têm o valor 0. A matriz tem SIZE*SIZE posições em que cada posição pode tomar um valor entre 0 e 255; supõe-se que SIZE é múltiplo de 4.

```
#include <stdio.h>
#include <pthread.h>
#define NTHREADS 4
#define SIZE 1024
unsigned char image[SIZE][SIZE];
// espaço para eventuais declararações em falta
int nzeros = 0;   int pzeros[NTHREADS];
void *worker(void * arg) {
        int i, j;
        int idx = (int) arg;
        pzeros[idx] = 0;
        for (i = idx; i < SIZE; i = i + NTHREADS)
                for(j = 0; j < size; j++)
                        if image[i][j] == (unsigned char) 0
                                pzeros[idx] ++;
        return NULL;
}
int main(int argc, char *argv[]){
  int i; pthread_id t[NTHREADS];
  // espaço para eventuais inicializações

   for (i=0; i < NTHREADS; i++)
           pthread_create( &t[i], NULL, worker, (void*)i );
   for (i=0; i < NTHREADS; i++)
           pthread_join( t[i], NULL );
//
   for (i = 0; i < NTHREADS; i++)
        nzeros = nzeros + pzeros[i];

   return 0;
}
```

Complete o código apresentado. Será valorizado código que minimize o número de operações de sincronização efectuadas.

Questão 5 – 2,0 valores Considere que se pretende construir um cliente e um servidor sequencial em que

- o cliente recebe do terminal o nome de um ficheiro de texto e envia-o, usando um datagrama UDP, para o servidor
- o servidor
  - envia um datagrama com o nº de bytes do ficheiro, que se supõe que existe
  - envia o ficheiro em tantos datagramas de tamanho BUFSiZE quanto o necessário
- o cliente vai mostrando no ecrã os bytes recebidos

Complete o código do servidor e do cliente

Cliente

```
#include "udp_comm.h"
#define BUFSIZE 1024
#define SERVER "Localhost"
#define SERVERPORT 1234
#define CLIENTPORT 1235
char request[BUFSIZE];
char reply[BUFSIZE];
struct sockaddr_in serv;
int sock;
int comp, nrecv;
int main(int argc, char *argv[]){
    sock = UDP_Open(CLIENTPORT);
    // teste de erro omitido
    fgets(request, BUFSIZE, stdin);
    UDP_FillSockAddr(SERVER, SERVERPORT);
    UDP_Write(sock, request, strlen(request+1));
    // recebe o comprimento do ficheiro
    UDP_Read(sock, &comp, sizeof(int));
    // recebe o ficheiro
    nread = 0;
    while (nread < comp) {
        nrecv = UDP_Read(sock, reply,
                         BUFSIZE);
        write(1, reply, nrecv);
        nread = nread + nrecv;
    }
    close(sock);
    return 0;
}
```

Servidor

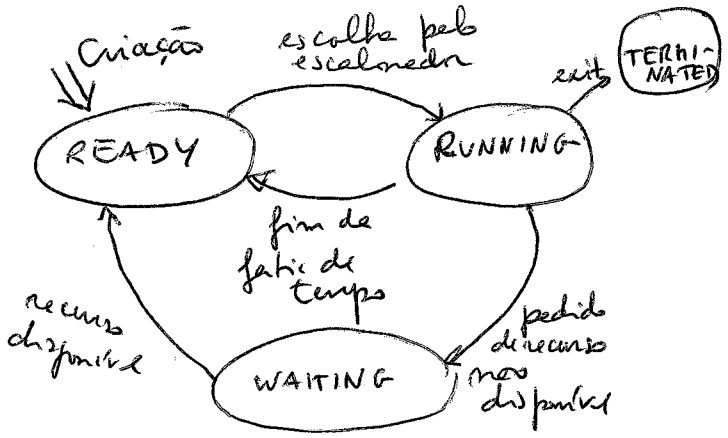
```
#include "udp_comm.h"
#define MAXW  32
#define BUFSIZE 1024
#define SERVERPORT 1234
char request [BUFSIZE];
char reply[BUFSIZE];
struct sockaddr_in cli;
int sock;
int f, nsent, nr;
struct stat fileInfo;
int main(int argc, char *argv[]){
    sock = UDP_Open(SERVERPORT);
    // teste de erro omitido
    while(1){
    //Receber o pedido
    UDP_Read(sock, request, BUFSIZE);
    // Preparar e enviar a resposta
    f = open(request, O_RDONLY);
    fstat(f, &fileInfo);
    UDP_Write(sock, &fileInfo.st_size,
              sizeof(off_t));
    nsent = 0;
    while(nsent < fileInfo.st_size){
        nr = read(f, reply, BUFSIZE);
        UDP_Write(sock, reply, nr);
        nsent = nsent + nr;
    }
    close(f);
}
```

Questão 6 – 1,5 valores Faça o diagrama de estados de um processo para os dois seguintes casos:

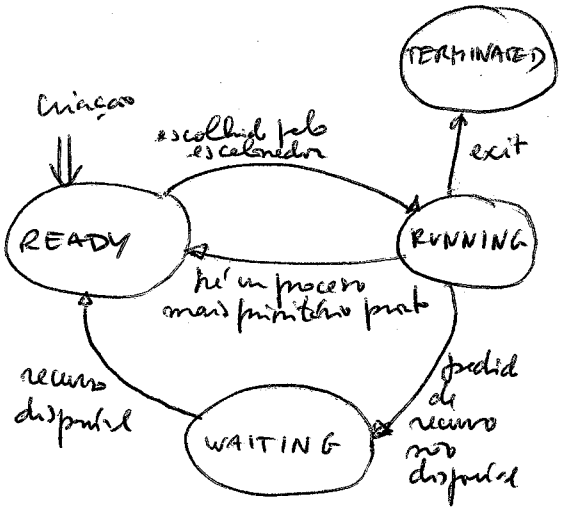
a) é utilizado o algoritmo de escalonamento MLFQ

b) é utilizado um algoritmo de escalonamento em que cada processo tem uma prioridade fixa e distinta de todos os outros processos; em cada momento, corre o processo com maior prioridade que estiver pronto

a)

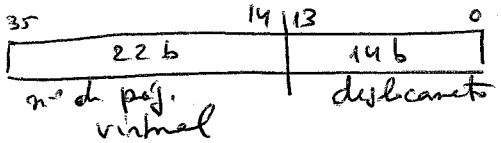


b)



Questão 7 – 1,0 valores

Considere um CPU com um endereço virtual com 36 bits e em que a memória central (RAM) é gerida usando páginas com 16 Kbytes (2^{14}). Qual o número de entradas da tabela de páginas de um processo? Justifique.



nº de entradas da tabela de páginas 2^{22}

Questão 8 (2.0 valores) Considere um SO que usa uma MMU baseada em páginas e com um bit de validade V. O SO e o hardware suportam paginação a pedido. Explique o que acontece quando o CPU referencia uma página virtual cujo bit V está a 0.

→ referência a uma página com o bit V a 0 provoca uma interrupção por falta de página

→ a rotina de tratamento de falta de página:
- obtém uma página física livre, recorrendo se necessário a um algoritmo de substituição de páginas
- bloqueia o processo, enquanto transfere a página do disco para RAM

→ a rotina de tratamento de interrupção fim de transferência de página actualiza a tabela de páginas e coloca o processo no estado READY

Questão 9 (2 valores) Considere o seguinte código, em que se supõe que o ficheiro executável "*xpto*" existe na directoria corrente.

...

```
char *args[MAXARGS];
int p,s;
p = fork();
if (p == 0) { // sem teste de erro
      args[0] = "./xpto";   args[1]=NULL;
      execvp("./xpto", args);
}
wait(&s);
```

O sistema operativo usa a técnica *copy on write*. Explique como é que é manipulada a tabela de páginas do filho, nos três seguintes momentos

a) na criação do processo

b) na execução da instrução `args[0] = "./xpto";`

c) na execução da chamada ao sistema *execvp*

a) cria a tabela de páginas do filho que é uma cópia integral da do pai. Todas as entradas da tabela de páginas do pai e do filho são marcadas com READ ONLY

b) a execução das instruções correspondentes vão provocar uma interrupção por acesso ilegal. O SO vai obter 1 página física nova que atribui ao filho. As páginas virtuais no pai e no filho são marcadas com READ/WRITE

c) A tabela de páginas do filho é destruída. É criada uma tabela de páginas com conteúdo e dimensão extraídos do ficheiro executável especificado na chamada ao sistema.

Questão 10 (2.25 valores) Considere que o periférico */dev/zeros* foi criado com os seguintes comandos executados pelo administrador de sistema.

```
mknod /dev/zeros c 298 0
chmod ugo+r /dev/zeros
```

Este dispositivo retorna tantos bytes a zero quantos os que são especificados no 3º parâmetro da chamada ao sistema *read*. Suponha que a estrutura *file_operations* associada ao periférico tem o seguinte conteúdo

```
static struct file_operations zeros_ops =
{
        .owner = THIS_MODULE,
        .open = zeros_open,
        .release = zeros_release,
        .read = zeros_read,
};
```

a) Descreva o que se passa quando é executada a chamada ao sistema.

```
int f = open( "/dev/zeros", O_RDONLY);
```

- é lido o dir. raiz para obter o i-node I de "/dev/zeros"
- lê-se o i-node I e obtém-se o major device number e o minor device number
- na entrada f da tabela de canais abertos é colocado um apontador para o i-node I em RAM
- é chamada a função zeros_open

b) Descreva as acções efectuadas por `zeros_read` quando é feita a chamada ao sistema

```
read( f, buf , 20);
```

- através de entrada f da tabela de canais abertos obtém o major device number
- invoca-se a função zeros_read que preenche o buffer passado pelo utilizador com bytes a 0 usando a função do Kernel copy_to_user.

Questão 11 (2.25 valores) Considere o seguinte comando do shell

```
touch ./x.log
```

Este comando procura na directoria corrente o ficheiro *x.log* e

- se *x.log* existe, muda a data do último acesso para a data corrente
- se *x.log* não existe, cria um ficheiro vazio com este nome

Suponha que o processo que faz as chamadas tem permissões para escrever na diretoria corrente. Descreva as alterações feitas na área de meta-dados do disco que suporta o sistema de ficheiros em que está o ficheiro *x.log*.

Supondo que o descritor do processo contém o i-node do directório corrente, este é lido e procurada a entrada "x.log"

- se esta entrada existir, o i-node é alterado em RAM na parte que diz respeito à data do último acesso

- se a entrada não existe
  - é obtido um i-node livre I que é marcado como ocupado
  - o i-node é preenchido, incluindo a data de criação e o comprimento (que é zero)
  - é criada uma entrada no directório corrente com nome "x.log" e i-node I

Questão 12 (1.5 valores) Considere um disco que depois de formatado tem 2^{24} blocos e em que cada bloco tem 2048 (2^{11}) bytes; os endereços dos blocos têm 32 bits (4 bytes).

Suponha que neste disco foi colocado um sistema de ficheiros que usa uma estratégia de atribuição de espaço em disco indexada, em que os blocos atribuídos a um ficheiro são definidos por 10 endereços de blocos:

- Os endereços 0 a 7 são endereços diretos
- O endereço 8 tem um endereço de um bloco que contém endereços de blocos.
- O endereço 9 tem um endereço de um bloco que contém endereços de blocos que, por sua vez, contêm endereços de blocos.

Diga, justificando, qual é o número máximo de blocos que, usando esta estratégia de atribuição, um ficheiro pode ter? É apenas necessário indicar a expressão usado para o cálculo.

$\frac{\text{Tamanho do bloco}}{\text{Tam de 1 endereço}} = 2^{11}/2^{2} = 2^{9}$ (512 endereços por bloco)

Tamanho máx $= 2^{11} * (8 + 512 + 512 * 512)$

8 ↑ end. directos; 512 ↑ nº de end. num bloco (indirect simples); 512 * 512 ↑ nº de endereços em 512 blocos (indirect dupla)

Questão 13 (1.5 valores) Suponha um sistema de ficheiros organizado da forma habitual no UNIX/Linux. Considere a execução do comando shell *rm xpto.txt*. em que supõe que *xpto.txt* existe e que o utilizador que executa o shell tem permissões para remover o ficheiro. Admita ainda que não há nenhum *hard link* a apontar para o i-node de *xpto.txt*.

Diga que blocos da área de metadados são alterados e discuta, do ponto de vista da manutenção da consistência do sistema de ficheiros, a ordem pela qual devem ser feitas as alterações nesses blocos.

Para remover o ficheiro xpto.txt é preciso começar por obter o seu i-node I

Caso 1) Contador de referências maior do que 1

1.1 é removida a directoria ~~correcta~~

1.2 é actualizado o contador de referências do i-node

Caso 2) Contador de referências igual a 1

2.1 é removida a entrada na directoria

2.2. o i-node I é dado como livre

2.3 os blocos referenciados por I são declarados como livres

No caso 1) a ordem indicada para a melhor, porque se 1.1 for feito e 1.2 não, um verificador de consistência resolve o problema decrementando o contador de referências no i-node

No caso 2) a ordem também para a indicada. Se só 2.1 for feito, o verificador vai alocar o ficheiro na dir "lost + found". Se 2.1 e 2.2 forem feitos e 2.3 não, o verificador de consistência vai libertar os blocos não referenciados por nenhum i-node