isec Engenharia

Sistemas Operativos

2021/22

DEIS – Engenharia Informática

Prova escrita
Parte I

22 / Fev / 2022

Duração **60 minutos**

Sem consulta

Parte 1

1. (10%) Os sistemas operativos com arquitetura *micro-kernel*, ou cliente servidor, têm uma característica saliente que lhe é apontada como vantajosa. Identifique essa característica e explique por que é que é uma vantagem (qual é por que é que é uma vantagem). A sua resposta deve mesmo incluir uma explicação técnica.

2. (12.5%) Assuma que está a desenvolver um sistema operativo para o processador 8086 (é aquele que estudou nas aulas de TAC do primeiro ano). Este processador tem as seguintes características: endereçamento: real; modos de execução/privilégio: único; capacidade de interrupções: *traps*, interrupções e *excepções*.
   O seu sistema conseguirá ter as características desejáveis que se encontram em sistemas modernos? Se achar que sim, justifique; se achar que não, identifique 3 características e para cada uma explique e justifique.

3. (12.5%) A tabela à direita descreve a totalidade do espaço de endereçamento de um processo num sistema com memória paginada em que cada página tem 1Kb. O sistema usa o algoritmo de substituição NRU.

	Base	Prot	Nivel	P	R	M
0	8192	rw-	n.a.	1	0	1
1	4096	rw-	n.a.	0	1	0
2	1024	rw-	n.a.	1	1	0
3	2048	rw-	n.a.	0	0	0

   a) Qual o endereço real usado quando o processo escreve a letra 'A' num caracter armazenado no endereço virtual 1030? Descreva e explique tudo o que acontece na sequência desta operação de escrita.

   b) Após a alínea a), o sistema determina que é preciso remover uma página a este processo. Explique, justificando, qual é a página removida.

4. (15%) Existe um programa com várias *threads*. Duas das *threads* ("FAZ") produzem informação que é colocada numa matriz. Uma terceira *thread* "USA" vai buscar essa informação e imprime-a. São 20 elementos de informação ao todo. O código abaixo mostra a implementação já existente (apenas são mostrados os detalhes relevantes)

Dados acessíveis a todo o processo

```
typedef struct {
   // ... etc.
} Info;

Info buffer[20];
int poe = 0;
int tira = 0;

pthread_mutex_t F;
pthread_mutex_t U;

// inicializa mutexes
```

Código das *thread* FAZ

```
// etc. não relacionado

Info faz;
while (poe < 20) {
  pthread_mutex_lock(& F);
  faz = produzInfo();
  pthread_mutex_unlock(& F);
  buffer[poe] = faz;
  ++poe;
}

// etc. não relacionado
```

Código da *thread* USA

```
// etc. não relacionado
Info usa;
while (tira < 20) {
  if (tira < poe) {
    usa = buffer[tira];
    pthread_mutex_lock(& U);
    // imprime os dados de usa
    // no ecrã
    pthread_mutex_unlock(& U);
    ++tira;
  }
}
// etc. não relacionado
```

Identifique eventuais problemas que possam existir na solução proposta. Para cada problema que identificar deve explicar o que se trata, e no final apresentar a correção (código do género do que foi apresentado no enunciado).

Nota: Se estiver a pensar em apontar problemas típicos de IP ("falta um ;", "o typedef está errado" (não está), etc.), então o melhor é ocupar o seu tempo noutra pergunta.

isec Engenharia

Sistemas Operativos

2021/22

DEIS – Engenharia Informática

Prova escrita
Parte II
Época Recurso
22 / Fev / 2022

Respostas Duração **1h30 minutos** Com consulta

- **Pergunta 1 e 2 numa folha**
- **Pergunta 3 noutra folha**

1. Utilizando apenas uma linha de comandos de Unix e sem recorrer a ficheiros temporários:
   a) (4 %) Acrescente ao ficheiro *novatos.txt*, que já se encontra na sua *homedir*, o primeiro nome dos últimos doze utilizadores que foram adicionados ao sistema. A informação deve ser colocada por ordem alfabética crescente. A operação efetuada deve ser independente do local de onde executa os comandos.
   b) (4%) Apresente no ecrã o número (quantidade) de letras do maior ficheiro na pasta /tmp que tenha permissões "ler e executar" (escrever = tanto faz) para o dono, "ler e escrever" (executar tanto faz) para o grupo e sem qualquer permissão para todos os outros utilizadores. A correta realização desta tarefa não deve depender da pasta atual em que é executada a linha de comandos.

2. (23,5%) Pretende-se um sistema baseado em Unix para gerir um aeroporto, atribuindo aviões a controladores aéreos. O sistema é constituído por 3 programas: ***avião*** (cada avião corresponde a uma execução deste programa), ***controlador*** (cada controlador aéreo executa este programa), e ***torre*** (existe apenas um processo a correr este programa).
   Os programas avião e controlador **já estão implementados** e o **foco da questão é o programa torre**. Os mecanismos de comunicação entre processos são os ***named pipes***.

   Quando um ***avião*** se aproxima do aeroporto, o processo que o representa irá contactar a ***torre***, fornecendo-lhe uma estrutura de dados com: a identificação da aeronave (15 letras) e o PID do processo. Os dados são enviados à ***torre*** através de um *named pipe* cujo nome se encontra na variável de ambiente NPA2TORRE.

   A ***torre*** deve responder ao **avião**, também através de um *named pipe*, com uma confirmação de que recebeu a mensagem. Após o envio da confirmação, encaminha os dados do avião a um dos vários controladores que existem e já se encontram a correr. O *named pipe* específico a cada controlador é encontrado da seguinte forma: "NPCONTR" concatenado com o número desse controlador. O primeiro é o controlador "1", e o número total de controladores é dado por argumento de linha de comandos da torre.

   Quando recebe a informação de um avião, o controlador fica encarrega desse avião. Cada controlador só pode lidar com 5 aviões de cada vez, e é importante que a torre tenha noção de quantos aviões estão a cada instante em cada controlador. Assim, cada controlador deve informar a torre, por *named pipe* (o nome é fixo = "NPC2TORRE") sempre que termina de controlar um avião. Se acontecer um avião não ter controlador na altura em que se dirige à torre, então a torre informa-o para regressar ao ponto de origem.
   A estratégia escolha do controlador para gerir um avião não é importante: pode ser o menos ocupado, o primeiro que não estiver totalmente ocupado, etc.

   A ***torre*** deve imprimir no ecrã, a cada 60 segundos, o número de aviões que estão atualmente a ser controlados por cada controlador.

   Implemente a ***torre***.

   Restrições de implementação
   - Não pode usar **nem sinais** nem variáveis globais
   - Deve evitar usar o mecanismo ***select***. Se o usar já não terá toda a cotação.

   NOTA: deve declarar todas as variáveis e estruturas de dados que utilizar. Não é necessário especificar *#includes*.

3. (18,5%) Escreva, em C, uma aplicação para Unix, de nome "cronometro", que permita cronometrar a execução de N comandos Unix. Trata-se dos vulgares comandos tais como *sort*, *head*, *cut*, *wc*, etc., e pode e deve assumir a partir destes exemplos a forma como estes programas comunicam. Pode assumir que nunca serão mais do que 10 comandos.
   A aplicação pretendida deverá executar cada um dos comandos, que lhe são indicados por linha de comandos, colocando-os a comunicar uns com os outros. O primeiro comando envia o output para o segundo, o segundo para o terceiro, e assim sucessivamente. Assim que o último comando termine, o "cronómetro" deve mostrar o tempo decorrido no monitor. Não deve aparecer qualquer output dos comandos no monitor. Caso a execução ultrapasse um determinado intervalo de tempo, especificado através da variável de ambiente TEMPMAX, o "cronómetro" deve terminar os comandos que ainda se encontrem em execução e avisar que fez isso.
   Pode assumir que existe e usar a função *int getSegundos()* que indica o número de segundos desde que a máquina foi iniciada.

   NOTA: deve declarar todas as variáveis e estruturas de dados que utilizar. Não é necessário especificar *#includes*.

   Exemplo de utilização

   $ ./cronometro who sort wc
   Cronómetro: a operação demorou 9 segundos