LICENCIATURA EM ENGENHARIA INFORMÁTICA

Sistemas Operativos

Teste[1]

14 de Junho de 2010 Duração: 2h00m

I

1 Relativamente a algoritmos de escalonamento, sabendo que o SRT – *shortest remaining time* leva tipicamente a menores tempos de espera na *ready queue* do que o RR – *round robin*, explique por que motivos este último acaba por ser um algoritmo muito popular.

2 Descreva em que consiste o fenómeno conhecido por *thrashing*. Explique como a interacção entre o sistema de memória virtual e um escalonador ingénuo que não tenha em conta o uso de memória pode levar à ocorrência de *thrashing*. Tem alguma sugestão concreta para conseguir esse escalonamento menos ingénuo? Consegue estimar o custo adicional dessa estratégia relativamente ao RR?

3 Recorde os seus conhecimentos de gestão de disco, em particular a tolerância a *bad sectors* e os objectivos dos vários tipos de RAID. Admita que se trata de um serviço que necessita de elevado desempenho dos discos (e.g. múltiplos streams de video de alta qualidade) e descreva a forma como configuraria este servidor. Pretende-se que discuta aspectos como o tipo de RAID, o escalonamento de pedidos de transferência de disco, substituição de blocos estragados, etc..

II

Considere o programa `getlfeed` que recebe como argumento único uma URL e imprime no seu stdout uma lista de notícias, uma em cada linha. Devido aos atrasos na rede, este programa demora um tempo considerável. Pretende-se escrever um comando `getallfeeds` que recebe como argumentos uma lista de URLs, imprimindo para o seu stdout todas as notícias dessas feeds que não contenham a palavra "apple". Para acelerar o resultado espera-se que vários `getlfeed` corram ao mesmo tempo. Para não sobrecarregar a rede, não devem correr mais de 10 `getlfeed` ao mesmo tempo.

III

Codifique um programa `pipeflow` que recebe como argumentos o nome de dois outros programas (ex: "pipeflow ls wc"). O programa deve correr os dois comandos, injectando o output do primeiro no input do segundo, e medir o fluxo nessa ligação. O programa deve a cada segundo mostrar no seu stdout quantos bytes foram transmitidos nesse segundo.

Protótipos de algumas funções e chamadas ao sistema relevantes

Processos

```
• pid_t fork(void);
• void exit(int status);
• pid_t wait(int *status);
• pid_t waitpid(pid_t pid, int *status, int
  options);
• WIFEXITED(status);
• WEXITSTATUS(status);
• int execlp(const char *file, const char
  *arg, ...);
• int execvp(const char *file, char *const
  argv[]);
• int execve(const char *file, char *const
  argv[], char *const envp[]);
```

Sinais

```
• void (*signal(int signum, void
  (*handler)(int)))(int);
• int kill(pid_t pid, int signum);
• int alarm(int seconds);
• int pause(void);
```

Sistema de Ficheiros

```
• int open(const char *pathname, int flags,
  mode_t mode);
• int creat(const char *pathname, mode_t
  mode);
• int close(int fd);
• int read(int fd, void *buf, size_t count);
• int write(int fd, const void *buf, size_t
  count);
• int pipe(int filedes[2]);
• int dup(int oldfd);
• int dup2(int oldfd, int newfd);
```

[1] Cotação — 8+6+6