MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

Sistemas Operativos

Exame de Recurso[1]

18 de Junho de 2019 — Duração: 2h

Por favor responda a cada um dos 3 grupos em folhas de teste separadas. Obrigado.

I

Responda a este grupo de forma muito sucinta: **não ultrapasse 15 linhas** em cada pergunta.

1 Um dos primeiros sistemas interactivos multi-utilizador usava *Multilevel Queues* com fatias de tempo de cerca de 100 ms que iam aumentando à medida que a prioridade descia. Uma vez por segundo transferiam-se todos os processos de cada queue para a queue imediatamente superior. Porém, com o passar dos anos, esta técnica conduziu a longos tempos de espera e foi abandonada. Procure explicar onde estaria o problema e como terá sido corrigido (mantendo MLQ).

2 Arquitecturas modernas suportam páginas de memória virtual que podem ser configuradas para terem tamanhos grandes (e.g., 2 MB).
(1) descreva vantagens e desvantagens destas *huges pages* face às páginas clássicas com 4 KB;
(2) dê um exemplo de uma estrutura de dados apropriada para ser alocada numa região de memória baseada neste tipo de páginas. Justifique;
(3) se fosse mapear em memória os ficheiros do trabalho prático, diga para cada um (artigos, strings, stocks, vendas) se deveria ou não considerar o uso de *huge pages*. Justifique.

II

Assuma: a existência de um programa `agrep`, que recebe dois argumentos, uma palavra e um nome de ficheiro, e produz no *stdout* todas as linhas do ficheiro contendo a palavra, cada uma escrita atomicamente num único `write`; que cada linha não excede os 100 bytes; a existência de uma função `readline`, com o mesmo protótipo de `read`.

Escreva um programa com dois argumentos, uma palavra e um nome de ficheiro de saída, que recebe pelo *stdin* uma lista de nomes de ficheiros, um por linha. Pretende-se que este programa, fazendo uso do `agrep`, concatene no ficheiro de saída todas as linhas dos ficheiros listados que contêm a palavra (sem preocupação com a ordem das linhas). Explore concorrência, mas limite a 10 o número máximo de processos a correr `agrep` em cada momento, e dê a cada execução de `agrep` um minuto para terminar, forçando a terminação se necessário.

III

Considere um programa que lê URLs do seu stdin (um por linha), descarrega a página Web correspondente e imprime os URLs nela contidos, cada um escrito atomicamente num único write. Para obter todos os URLs, sem repetições, contidos nas páginas dadas por uma lista de URLs, usa-se na shell em conjunto com o programa `uniq` da seguinte forma:

```
$ fetchurls < seedURLs.txt | uniq
```

Escreva um programa em C com a mesma funcionalidade e fazendo uso dos programas `fetchurls` e `uniq` mas que permita ter 8 processos a descarregar páginas concorrentemente. Assuma que cada URL não excede os 100 bytes e a existência de uma função `readline`, com o mesmo protótipo de `read`.

Algumas 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[]);

Sistema de Ficheiros

- int open(const char *pathname, int flags, 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);
- long lseek(int fd, long offset, int whence);
- int access(const char *pathname, int amode);
- int pipe(int filedes[2]);
- int dup(int oldfd);
- int dup2(int oldfd, int newfd);

Sinais

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

[1] Cotação: 6+7+7