MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Sistemas Operativos

Exame de Recurso

26 de Junho de 2017 — Duração: 2h

I

1 Ao criar um processo em Linux, é habitual encontrar o padrão seguinte:

```
if ((p = fork()) == 0) /* filho... */ else /* pai... */
```

Explique com rigor como é possível executar simultaneamente 2 processos "iguais" sem que o filho consiga aceder às variáveis do pai (e vice-versa).

2 Explique a necessidade do fork() acima ser uma *system call* e não uma função normal.

3 Discuta em que medida um sistema de Raid 5 permite recuperar informação adulterada num ataque de "ransomware" (ficheiros deliberadamente cifrados para forçar pagamento de resgate).

II

Escreva um programa, invocado da forma `controlador <p> <c>`, para processamento de linhas de texto produzidas concorrentemente por *p* processos, instâncias de um programa `produtor` (que se assume existir, invocado sem argumentos), por um de *c* processos, instâncias de um programa `consumidor` (que se assume existir, invocado sem argumentos). Em cada momento todas as linhas produzidas devem ser processadas por um mesmo consumidor. Com o tempo deve ser feita uma escolha rotativa do consumidor corrente, devendo ser escolhido o consumidor seguinte quando o processo `controlador` receber o sinal `SIGUSR1`.

III

Escreva um programa, invocado da forma `streamer <p1> <p2> ... <pn>`, que encadeia as saídas e entradas standard dos programas `p1` a `pn`. No caso de `streamer` receber um sinal `SIGUSR1`, os programas de deverão ser terminados assegurando que qualquer informação lida por `p1` é antes processada e impressa por `pn`.

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);

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Sistemas Operativos

Teste

8 de Junho de 2018 — Duração: 2h

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

I

1 Admita que um grande conjunto de dados que tem de ser processado em cadeia. Demonstre que neste caso a comunicação entre processos por *pipes anónimos* é muito mais eficiente e conveniente do que a utilização de ficheiros, sendo provável que também contribua para a redução de tempos de resposta.

2 Suponha que ao instalar um sistema operativo num computador (eg, linux, a ser usado nas aulas de SO) lhe aparece uma "caixa" que pede para escolher o Grau de Multiprogramação. Por omissão fica a 1 mas pode ajustar esse valor entre 1 e 999. Explique os critérios em que basearia a sua escolha e justifique o valor que escolheu.

3 Se um determinado programa demora a executar 5 minutos quando não há mais nada em execução num computador, qual o tempo médio de execução se forem executados simultaneamente 4 processos idênticos, no caso do escalonamento ser FIFO ou RR, respectivamente. Escolha uma fatia de tempo adequada ao computador em causa e apresente as contas que fez para chegar aos resultados. Que conclusões tira?

II

Considere um programa 'controlador' que deverá executar repetidamente um conjunto de programas especificados na sua linha de comando, até um destes terminar com código de saída de 0. Os programas deverão ser todos executados concorrentemente, em tentativas sucessivas. Sempre que numa tentativa nenhum dos processos terminar com código de saída de 0, a próxima tentativa tem como limite de tempo (em segundos) a soma dos códigos de saída. Findo esse limite de tempo, todos os processos, incluindo o controlador, devem ser terminados. A primeira tentativa não tem limite de tempo. Implemente o programa controlador recorrendo às primitivas de processos e sinais estudadas.

III

Escreva um programa, invocado da forma `controlador` p c, para processamento de linhas de texto produzidas concorrentemente por p processos, instâncias de um programa `produtor`, por c processos, instâncias de um programa `consumidor`, ambos invocados sem argumentos. Assuma que cada linha produzida começa por um dígito d, e faça com que o resto da linha seja processada pelo consumidor d, quando $0 \leq d < c$, ou descartada caso contrário.

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);

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Sistemas Operativos

Teste

26 de maio de 2017 — Duração: 2h

I

Responda às perguntas seguintes com *palavras suas* e de forma *sucinta*: Não exceda o limite de 10 a 15 linhas legíveis em cada.

1 Praticamente todos os dias surgem notícias de ataques a sistemas informáticos e da necessidade de protecção. Ora sendo o sistema operativo um gestor de recursos, também lhe cabe a tarefa de fornecer mecanismos para proteger esses recursos. Escolha três recursos estudados nas aulas teóricas e explique os respectivos mecanismos de protecção. Assuma que se trata de um sistema com Linux.

2 Ao programar e testar alguns exercícios propostos nos guiões das aulas práticas certamente se terá deparado com situações de *Deadlock*. Dê um ou mais exemplos **seus** (reais ou imaginados por si) que mostrem com precisão o problema e a forma de o evitar. Deve dar uma resposta muito concreta.

3 Leia com atenção o enunciado do Grupo II. Acha que para resolver esta questão seria boa ideia usar uma estratégia de escalonamento *ROUND-ROBIN* nos processos passados como argumentos (no exemplo, ls df find). Justifique a sua resposta. Mostre que percebeu a pergunta (e a resposta).

II

Pretende-se escrever o programa "paginas", que recebe como argumentos os nomes de vários executáveis e que faz com que o output desses programas apareça alternadamente, página por página, de 10 linhas cada uma. Por exemplo, correndo "paginas ls df find", deverão ser impressas 10 linhas do ls, depois 10 linhas do df, depois 10 linhas do find, depois 10 linhas do ls e assim sucessivamente até que os vários programas terminem. A ausência de output de um dos programas não deverá impedir a impressão das linhas dos outros.

III

Considere que se pretende fazer uma pesquisa concorrente para determinar a existência de determinado padrão em linhas de texto, recebidas no stdin. Assuma a existência dois programas, `filtro` e `existe` que, encadeados, permitem fazer a pesquisa. O primeiro descarta as casos mais óbvios que lhe chegam ao seu stdin, deixando passar os restantes para o seu stdout. O segundo termina com valor de exit de 1 caso receba uma linha com o padrão no seu stdin, ou com 0, caso isso não aconteça, até EOF. Escreva um programa que receba um número n como argumento, e faça a pesquisa concorrente utilizando n pares de processos a executar `filtro` e `existe`. O programa deverá imprimir o resultado, logo que possível, e terminar os processos que já não estejam a faer processamento relevante.

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);

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Sistemas Operativos

Exame de Recurso

29 de Junho de 2018 — Duração: 2h

I

1 (max 20 linhas) Suponha que é administrador/a de um sistema com utilizadores a queixarem-se que "de repente o sistema ficou lento", tendo de imediato avaliado a carga do processador. Que atitude(s) tomaria no caso de constatar que a carga média dos últimos 5 minutos se tinha mantido estável mas nos últimos 30 segundos tinha subido até perto do valor máximo? Que faria se, pelo contrário, a carga tivesse caído para 10 por cento da carga máxima?

2 (max 20 linhas) Suponha que, ao desenvolver uma aplicação informática, alguém lhe sugere a utilização de *arrays dinâmicos* para ler de ficheiro(s) a informação que é depois processada em memória dinâmica e escrita novamente em ficheiros. Baseando-se APENAS nos conhecimentos adquiridos em Sistemas Operativos mostre que esta abordagem pode conduzir a problemas de eficiência e elevado tempo de resposta, podendo mesmo não passar nos testes finais. Note que esta não é uma pergunta de *pipelines*. É proibido mencionar essa palavra!

II

Pretende-se escrever um programa que deverá executar concorrentemente um conjunto de programas especificados na sua linha de comando. O programa deverá avaliar a cada segundo a carga da máquina, acessível no ficheiro `/proc/load`, o qual contém apenas um número inteiro (em texto). A execução dos processos deverá ser suspendida enquanto a carga for superior a 100. O programa deverá terminar assim que detectar que todos os processos terminaram, com sucesso apenas caso todos tenham terminado com sucesso. Implemente o programa recorrendo a primitivas de processos e sinais. Sugestão: faça uso dos sinais SIGSTOP e SIGCONT.

III

Considere a existência de um programa `termometro` que recebe um inteiro como argumento, indicando o dispositivo cuja temperatura deverá monitorizar. Este programa escreve para o *standard output* 3 digitos com o valor da temperatura do dispositivo **apenas** quando esta varia. Implemente um programa `monitor` que é invocado com um número arbitrário de argumentos inteiros que correspondem aos dispositivos a monitorizar. O programa `monitor` deverá executar concorrentemente (e com o devido argumento) tantas instâncias de `termometro` quantos os dispositivos a monitorizar. Além disso, deverá também executar o programa existente `alarme`, sem argumentos. Os valores de cada `termometro` devem ser passados ao *standard input* de `alarme` na forma `[d t]` em que `d` é o número do dispositivo e `t` o valor da sua temperatura. Em caso de uma situação anomala, o programa `alarme` termina, e todos os processos `termometro` e o próprio `monitor` deverão também terminar.

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);