LICENCIATURA EM ENGENHARIA INFORMÁTICA
LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

Sistemas Operativos

Exame - Época de Recurso[1]

3 de Julho de 2015 Duração: 2h00m

I

1 Suponha que existe uma impressora num sistema e que para imprimir basta os utilizadores usarem o comando `lpr2015 <ficheiro>`, cujo esqueleto se apresenta de seguida. Note que a função `imprime` já existe, basta invocá-la para que o ficheiro seja enviado para a impressora, mas tenha em atenção que terá de se preocupar com tentativas de impressão ao mesmo tempo.

```
shared ...              // vars partilhadas
main(ficheiro) {        // não se preocupe com argc e argv
     // ...
     imprime (ficheiro)    // despeja o ficheiro completo
     // ...
}
```

O que se pede é que pense numa forma de desligar a impressora de uma forma "civilizada". Para isso, deverá conceber outro comando (o nome é irrelevante mas pode pensar que se chama `lpr_off`, comando esse que é executado por um administrador de sistemas e que tem o efeito de, a partir desse momento, forçar o `lpr2015` a recusar-se a aceitar mais pedidos de impressão. Por seu lado, este `lpr_off` deverá esperar até que a fila de pedidos ainda por imprimir ou em impressão esteja vazia e terminar depois de produzir uma mensagem adequada para o administrador.

Em resumo, deve completar o `lpr2015 <ficheiro>` e escrever de raíz o `lpr_off`. Pense com calma, porque tem *muito pouco* código para escrever. Como estamos a trabalhar com memória partilhada, pede-se que use `cria_semaforo`, `P` e `V` para coordenar a concorrência entre os processos.

2 Normalmente o sistema operativo mantém uma lista de apontadores para o chamado *process register block* (PRB) de cada processo marcado como READY e daí retira o próximo processo a executar usando uma *estratégia de escalonamento*. *Round-robin* (RR) é uma das mais conhecidas e têm sido propostas variantes que permitem que um processo possa ter várias entradas na lista READY. Que consequências teria esta modificação? Em que circunstâncias seria preferível face ao RR tradicional? Fale da carga do sistema!

v.p.p.f.

---

[1] Cotação — 8+6+6

II

Considere um programa chamado `avariado` que por vezes bloqueia durante a execução, sem conseguir realizar a sua função e sem terminar. Sabemos que, enquanto funciona corretamente, o `avariado` envia o sinal SIGUSR1 ao processo pai pelo menos uma vez por segundo. Escreva um programa `remendo` que (re)executa o `avariado` as vezes necessárias até que ele conclua com sucesso. Não permita que o avariado fique bloqueado por mais de 5 segundos de cada vez.

III

Considere que pretende testar um programa `calculo` (já existente), que repetidamente lê uma linha do standard input e produz outra no standard output. Implemente um programa `testador` que deverá receber no seu standard input uma sequência de pares de strings (input e output esperado). Segue-se um exemplo de um possível input deste programa:

```
1+1
2
2*3+1
7
```

Perante o exemplo acima, a string `1+1` deve ser enviada ao programa "calculo" e este deve responder com `2`, e assim por diante. Casa se verifique alguma divergência face à resposta esperada, tanto o programa `testador` como o `calculo` deverão ser imediatamente terminados, e retornado um código de saída de correspondente a insucesso.

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);`
- `int mkfifo(const char *path, mode_t mode);`