LICENCIATURA EM ENGENHARIA DE SISTEMAS E INFORMÁTICA

Sistemas Operativos II

Recurso[1]

22 de Julho de 2004 Duração: 2h30m

I

Suponha que num sistema multi-utilizador como o Linux ou Windows XP lhe é pedido que coordene os acessos eventualmente concorrentes a um determinado recurso. O recurso em si não é particularmente relevante (imagine que pode ser um modem, uma impressora, um sensor de temperatura ou mesmo um ficheiro partilhado); o importante é que vários *processos* pertencentes a diferentes utilizadores poderão querer ter acesso ao recurso "ao mesmo tempo". Para simplificar, pode assumir que se pretende acesso exclusivo ao recurso.

1 Para o caso concreto aqui indicado, compare a utilização de semáforos e de variáveis do tipo condição como mecanismo capaz de garantir a exclusão mútua no acesso ao recurso partilhado.

2 Ainda no contexto da questão anterior, compare a solução *self-scheduling* com uma solução baseada na utilização de um Gestor do recurso partilhado. Assegure-se que explica convenientemente em que consiste cada tipo de solução.

3 Estude agora o enunciado da pergunta II. Se lhe fosse pedido que resolvesse este exercício com semáforos, a solução seria mais simples ou mais complicada? Porquê?

II

Suponha uma simulação da seguinte situação: pessoas partem do ponto A onde têm que apanhar um barco para o ponto B; aí têm que atravessar uma ponte de corda até ao ponto C. Codifique as seguintes funções, para serem invocadas por threads que representam pessoas:

- `apanha_barco()`, invocada no ponto A, que bloqueia até a pessoa poder embarcar;
- `inicia_travessia()`, invocada no ponto B, que bloqueia até a pessoa poder atravessar a ponte;
- `fim_travessia()`, invocada no ponto C, para assinalar fim de travessia da ponte.

A simulação está condicionada pelo seguinte: para rentabilizar o barco, este só parte quando estiverem `LOTACAO` pessoas à espera de embarcar; só pode estar uma pessoa de cada vez a atravessar a ponte de corda; o barco depois de chegar ao ponto B, só volta ao ponto A depois de todas as pessoas que levou terem atravessado a ponte de corda.

III

Pretende-se desenvolver um sistema distribuído de busca de ficheiros *peer-to-peer*. Para o efeito deverá desenvolver o código de um processo mediador que se interpõe entre os clientes e os vários servidores disponiveis. Na invocação, é dado a conhecer ao mediador o conjunto de servidores (eg. vector com ips). Para cada pesquisa dos clientes, o mediador deverá *concorrentemente* reencaminhá-la para todos os servidores e, posteriormente, enviar ao cliente todas as respostas que obtiver dos servidores. Após enviar a última resposta, o mediador deverá encerrar a ligação com o cliente. Para a resolução do exercício considere que o pedido dos clientes poderá ser uma qualquer sequência de caracteres terminada por "new line"e que todos os servidores encerram a ligação no fim da resposta.

Protótipos das chamadas ao sistema relevantes

Sockets BSD

- `int socket(int domain, int type, int protocol);`
- `int bind(int s, const struct sockaddr *name, int namelen);`
- `int listen(int s, int backlog);`
- `int accept(int s, struct sockaddr *addr, int *addrlen);`
- `int connect(int s, struct sockaddr *name, int namelen);`
- `int close(int s);`
- `int read(int fd, char *buffer, size_t len);`
- `int write(int fd, const char *buffer, size_t leng);`
- `u_long htonl(u_long hostlong);`
- `u_short htons(u_short hostshort);`
- `u_long ntohl(u_long netlong);`
- `u_short ntohs(u_short netshort);`
- `unsigned long inet_addr(const char *cp);`
- `struct sockaddr_in { short sin_family; u_short sin_port; struct in_addr sin_addr; char sin_zero[8]; };`

Threads POSIX

- `int pthread_create(pthread_t *threadid, const pthread_attr_t *attr, void *(*start_func)(void *), void *arg);`

[1] Cotação — 6+7+7

- `void pthread_exit(void *status);`
- `int pthread_join(pthread_t threadid, void **status);`
- `int pthread_detach(pthread_t threadid);`
- `int pthread_mutex_init(pthread_mutex_t *mp, const pthread_mutexattr_t *attr);`
- `int pthread_mutex_lock(pthread_mutex_t *mp);`
- `int pthread_mutex_unlock(pthread_mutex_t *mp);`
- `int pthread_mutex_destroy(pthread_mutex_t *mp);`
- `int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);`
- `int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);`
- `int pthread_cond_signal(pthread_cond_t *cond);`
- `int pthread_cond_broadcast(pthread_cond_t *cond);`