LICENCIATURA EM ENGENHARIA DE SISTEMAS E INFORMÁTICA

Sistemas Operativos II

Segunda chamada[1]

8 de Julho de 2004 Duração: 2h30m

I

1 Suponha que lhe pediram para especificar as características de uma máquina capaz de suportar o sistema de ”e-learning”de uma grande instituição. O que propõe, nomeadamente em relação ao sistema de ficheiros e à gestão do(s) disco(s)? Justifique.

2 Proponha uma estratégia de escalonamento de pedidos de transferência de disco adequada a um sistema com “mirror” de discos. Explique em que medida o ”mirroring”afecta ou não este escalonamento.

II

Considere um sistema com N servidores HTTP cujos endereços IP constam num vector serv[N]. Todos os servidores escutam na porta 8080. Pretende-se desenvolver um serviço de equilíbrio de carga que medeie a interacção dos clientes com os servidores HTTP de forma a que os pedidos sejam distribuídos de igual forma pelos N servidores.

1 Escreva o código do serviço de equilíbrio de carga. Um pedido HTTP corresponde a uma sequência de caracteres terminada por “newline”. Após responder, o servidor fecha a ligação com o cliente.

2 Suponha que é necessário manter a “afinidade” entre ligações, isto é, garantir que para ligações consecutivas de um cliente, os pedidos são tratados pelo mesmo servidor. Que alterações precisaria de introduzir no código anterior?

III

Considere um device-driver de disco que oferece a seguinte função para escrever um bloco:

```
void write(int block, char* data);
```

Para utilizar este device-driver a partir de um sistema de ficheiros multi-threaded é necessário garantir que a função não é invocada concorrentemente por mais do que um thread para o mesmo bloco. Para optimizar o desempenho, pretende-se que um máximo de MAX blocos possam ser escritos concorrentemente. Para o efeito escreva a função:

```
void threaded_write(int block, char* data);
```

utilizando primitivas de threads POSIX.

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);`
- `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);`

---

[1] Cotação — 6+7+7