12 Mars 1991

Examen DESS

de Génie Logiciel

UNIX-C

durée: 2 heures
tout document autorisé

Notation:

Exercice	:	a.	1	point
		b.	1	point
		c.	1	point
		d.	2	points
Problème	:	a.	3	points
		b.	8	points
		c.	4	points

Exercice

Présentation

On imagine un démon qui reçoit des ordres de clients **(éventuellement depuis d'autres machines)**, ces ordres étant des demandes de traçage de lignes sous X-Window.

Le **main** du démon est le suivant:

```
#include <X11/Xlib.h>
#include <stdio.h>

 Display *dpy;
 Window   w;
 GC gc;

 main(argc,argv)
/***************/

 int    argc;
 char **argv;

{
 int s;
 int connection;

 init_X_window();
 s=create_service();
 while (1)
   {
    connection=accept_request(s);
    give_service(connection);
    close(connection);
   }
 XCloseDisplay(dpy);
}
```

Questions:

a.: **écrire la fonction give_service(connection) qui reçoit un ordre et trace la ligne demandée.**

b.: **écrire la fonction create_service();**

c.: **écrire la fonction init_X_window();**

d.: **écrire une version simple d'un client.**

Problème

Présentation

On souhaite réaliser un démon auquel des clients sont connectés en permanence. Ces derniers lui envoient des lignes de texte qu'il a pour rôle d'afficher.

En ce qui concerne le démon, une solution naïve serait d'écrire:

```
#define NB_CLIENTS 2

 main(argc,argv)
/***************/
 int    argc;
 char **argv;

{
 int s;
 int connection[NB_CLIENTS];
 int i;
 char c;

 s=create_service(NB_CLIENTS);
 for (i=0;i<NB_CLIENTS;i++) connection[i]=accept_request(s);
 while (1)
    {
     for (i=0;i<NB_CLIENTS;i++)
        {
         do
           {
            read(connection[i],&c,1);
            write(1,&c,1);
           }
         while (c != '\n');
        }
    }
}
```

On est alors confronté au problème suivant: le démon doit recevoir de chacun des clients successivement et donc si un client n'envoie aucun message le démon se trouve bloqué, en attente de réception de ce client.

Questions:

a. A quoi sert, à votre avis, le paramètre passé à la fonction create_service ?
b. En utilisant select (c.f. documentation ci-jointe) présenter une solution simple permettant de résoudre le problème de blocage éventuel du démon.

c. La solution proposée en b. assure-t-elle une équité entre les clients ? (justifiez votre réponse). Si cela n'est pas le cas, proposez une solution pour assurer cette équité.

Documentation jointe pages 6 à 9:

- manuel de select (pages 6 à 8)

- manuel de getdtablesize (page 9)

Documentation pour le problème

```
NAME
     select - synchronous I/O multiplexing

SYNOPSIS
     #include <sys/types.h>
     #include <sys/time.h>

     int select (width, readfds, writefds, exceptfds, timeout)
     int width;
     fd_set *readfds, *writefds, *exceptfds;
     struct timeval *timeout;

     FD_SET (fd, &fdset)
     FD_CLR (fd, &fdset)
     FD_ISSET (fd, &fdset)
     FD_ZERO (&fdset)
     int fd;
     fd_set fdset;

DESCRIPTION
     select() examines the I/O descriptor  sets  whose  addresses
     are  passed  in  readfds,  writefds, and exceptfds to see if
     some of their descriptors are ready for reading,  ready  for
     writing, or have an exceptional condition pending.  width is
     the number of bits to be  checked  in  each  bit  mask  that
     represent  a file descriptor; the descriptors from 0 through
     width-1 in the  descriptor  sets  are  examined.   Typically
     width  has  the  value  returned by getdtablesize(2) for the
     maximum number of file  descriptors.   On  return,  select()
     replaces  the  given descriptor sets with subsets consisting
     of those descriptors that are ready for the requested opera-
     tion.  The total number of ready descriptors in all the sets
     is returned.

     The descriptor sets are stored as bit fields  in  arrays  of
     integers.   The following macros are provided for manipulat-
     ing such descriptor sets:  FD_ZERO  (&fdset)  initializes  a
```

descriptor set fdset to the null set. FD_SET(fd, &fdset) includes a particular descriptor fd in fdset. FD_CLR(fd, &fdset) removes fd from fdset. FD_ISSET(fd, &fdset) is nonzero if fd is a member of fdset, zero otherwise. The behavior of these macros is undefined if a descriptor value is less than zero or greater than or equal to FD_SETSIZE, which is normally at least equal to the maximum number of descriptors supported by the system.

If timeout is not a NULL pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a NULL pointer, the select blocks indefinitely. To effect a poll, the timeout argument should be a non-NULL pointer, pointing to a zero-valued timeval structure.

Any of readfds, writefds, and exceptfds may be given as NULL pointers if no descriptors are of interest.

Selecting true for reading on a socket descriptor upon which a listen (2) call has been performed indicates that a subsequent accept(2) call on that descriptor will not block.

RETURN VALUE

select() returns the number of ready descriptors that are contained in the descriptor sets, or -1 if an error occurred. If the time limit expires then select() returns 0. If select() returns with an error, including one due to an interrupted call, the descriptor sets will be unmodified.

ERRORS

An error return from select() indicates:

EBADF One of the descriptor sets specified an invalid descriptor.

EINTR A signal was delivered before any of the selected events occurred, or before the time limit expired.

EINVAL — A component of the pointed-to time limit is outside the acceptable range: t_sec must be between 0 and 10^8, inclusive. t_usec must be greater-than or equal to 0, and less than 10^6.

EFAULT — One of the pointers given in the call referred to a non-existent portion of the process' address space.

SEE ALSO

accept(2), connect(2), getdtablesize(2), gettimeofday(2), listen(2), read(2V), recv(2), send(2), write(2V)

BUGS

Although the provision of getdtablesize(2) was intended to allow user programs to be written independent of the kernel limit on the number of open files, the dimension of a sufficiently large bit field for select remains a problem. The default size FD_SETSIZE (currently 256) is somewhat larger than the current kernel limit to the number of open files. However, in order to accommodate programs which might potentially use a larger number of open files with select, it is possible to increase this size within a program by providing a larger definition of FD_SETSIZE before the inclusion of /usr/include/sys/types.h.

select() should probably return the time remaining from the original timeout, if any, by modifying the time value in place. This may be implemented in future versions of the system. Thus, it is unwise to assume that the timeout pointer will be unmodified by the select() call.

NAME

getdtablesize - get descriptor table size

SYNOPSIS

getdtablesize()

DESCRIPTION

Each process has a fixed size descriptor table, which is guaranteed to have at least 20 slots. The entries in the descriptor table are numbered with small integers starting at 0. The call getdtablesize() returns the size of this table.

SEE ALSO

close(2), dup(2), open(2V)