UMEÅ UNIVERSITET
Institutionen för datavetenskap
Gunnar Gunnarsson, Tomas Halvarsson

Tentamen Datavetenskap

Kurs: Systemprogrammering, 5p
Tentamensdatum: 23 augusti 1999 kl 9-15
Hjälpmedel: *A book on C* / *Vägen till C*
Examinator: Gunnar Gunnarsson (gunnar@cs.umu.se)

Personnummer: _____

Namn: _____

Namnteckning: _____

Kurs:
- ❑ Systemprogrammering för C-programmet
- ❑ Datavetenskap B mom 1 (Systemprogrammering)
- ❑ Systemprogrammering för ingenjörer

Maxpoäng: 40 poäng
Betygsgränser: 20 poäng (betyg 3), 26 poäng (betyg 4), 32 poäng (betyg 5)
respektive 20 poäng (godkänt), 30 poäng (väl godkänt)

Kryssa för de uppgifter som du har löst/svarat på.

Uppgift	1	2	3	4	5	6	7	8	9	10	Totalt
Löst											
Maxpoäng	4	4	4	6	2	3	2	5	6	4	40
Poäng											

Betyg: _____ Signatur: _____ Inlämningstid: _____

Kom ihåg att även ofullständigt lösta uppgifter kan ge poäng. Börja varje uppgift på ett nytt blad. Skriv endast på framsidan. Skriv ditt namn och uppgiftens nummer på varje blad. Observera att uppgifterna inte är ordnade i stigande svårighetsgrad. Tre sidor med funktionsprototyper finns bifogade. Använd också boken om du har den med dig, den innehåller i vissa fall mer information.

Uttryck dig **mycket tydligt**. Alla **otydligheter** tolkas till din **nackdel**.

Uppgift 1 (4 p)

Förklara kortfattat och mycket tydligt innebörden av följande termer i Unix-sammanhang:

a) fildeskriptor
b) signalhanterare
c) I/O-multiplexering
d) standard output
e) interpreter-fil
f) FIFO (named pipe)
g) barnprocess
h) curses

Uppgift 2 (4 p)

a) Beskriv med ord vilken typ som skapas av följande deklaration:

```
typedef char (*ath)(int *);
```

b) Hur många varv körs nedanstående slinga?

```
i = 10;
while (--i > 5) call_function();
```

c) Visa hur man i C deklarerar en tvådimensionell array med 5x5 element, där elementtypen är "pekare till float".

d) Vad skrivs ut av nedanstående sats?

```
printf("Siffra: %d.\n", (a=8, b=3));
```

Uppgift 3 (4 p)

När två processer ska kommunicera med varandra kan de använda sig av många olika metoder. En av de enklaste metoderna, som använts på laborationerna, är vanliga enkelriktade pipor som skapas med `pipe()`. Nämn två andra metoder och förklara i grova drag hur de fungerar och vilka för- och nackdelar de har.

Uppgift 4 (6 p)

Nedanstående program ska ta ett antal filer som argument, och för varje fil skriva ut det största talet på första raden i filen. (Varje fil antas innehålla två heltal på första raden.) Koden innehåller dock ett antal logiska och syntaktiska fel, som gör att programmet inte kommer att fungera som avsett när det kompileras och körs på de maskiner som använts på kursen. Vilka är dessa fel?

Ange minst fyra fel, och ge en förklaring till varför de är fel, och vad som händer. När du förklarar konsekvenserna av ett fel, anta att bara det felet finns i programmet, dvs att de andra felen är åtgärdade. Observera att ingen enskild rad i källkoden innehåller mer än ett fel!

```
#include <stdio.h>
#include <stdlib.h>

#define MAX(a,b) ((a) > (b) ? (a) : (b));

int main (int argc, char *argv[]) {
    int number1, number2;
    int i;
    FILE *fp;

    for (i = 1; i < argc; i++) {
        fp = fopen(argv[i], "r");
        fscanf(fp, "%d %d", number1, number2);
        printf("Den största siffran på första raden i filen %s "
               "är %d.\n", argv[i], MAX(number1, number2));
    }
    return 0;
}
```

Uppgift 5 (1+1=2 p)

Ett modernt Unix-filsystem stöder både hårda länkar och mjuka (symboliska) länkar. Båda typerna kan skapas med kommandot `ln`.

a) Nämn en grundläggande skillnad mellan hårda och mjuka länkar.

b) När man gör en rekursiv traversering av ett filsystem (som på en av laborationerna) bör man inte följa symboliska länkar till kataloger. Varför inte?

Uppgift 6 (3 p)

Följande meningar diskuterar ämnen som har med systemprogrammering att göra. Din uppgift är att tala om vilka ord symbolerna A, B, C, X, Y och Z står för. Varje symbol motsvarar exakt ett ord. Om samma symbol förekommer flera gånger motsvarar den samma ord varje gång.

*”När en **X** levereras till en process tas den omhand av en **Y**, om en sådan finns installerad. En **X** som genererats men ännu inte levererats sägs vara en **Z X**.”*

*”När man anropar en **A** måste man tänka på att testa dess **B**. Anledningen är att en **A** kan misslyckas, och om man då inte hanterar felet på ett bra sätt kan stora problem uppstå. För att få mer detaljerad information om vad som gick fel i en **A** kan man avläsa den globala variabeln **C**.”*

Uppgift 7 (2 p)

a) Nämn en viktig skillnad mellan vanliga pipor (sådana som skapas med `pipe()`), och FIFO:s (sådana som skapas med `mkfifo()`).

b) Nedanstående shellskript, `xcd`, är tänkt att användas för att snabbt kunna gå till vissa vanliga kataloger:

```
#!/usr/bin/sh
case $* in
  1) ndir="/usr/local/bin" ;;
  2) ndir="/usr/include" ;;
  *) ndir="/Home/staff/gunnar/src" ;;
esac
echo Changing to directory $ndir...
cd $ndir
```

När det provkörs visar det sig att det inte fungerar som avsett:

```
~/exp@peppar% xcd 1
Changing to directory /usr/local/bin...
~/exp@peppar%
```

Det skriver ut alltså rätt katalognamn, men `cd`-kommandot verkar inte ge någon effekt. Vad beror detta på?

Uppgift 8 (3+2=5 p)

a) Två processer, A och B, startas av en gemensam förälder. Föräldern har använt `pipe()` för att skapa en pipa P, som A och B ska använda för att överföra data mellan varandra. Pseudokod för båda processerna finns nedan. Kommer detta att fungera som avsett eller ej? Förklara. (Observera att du kan få poäng oavsett om du svarar ja eller nej på frågan! Det är din motivering som räknas.)

```
A:
  Skriv ett block med 8192 bytes till P
  Läs 4096 bytes från P
  Behandla de 4096 lästa byten på något sätt
  Avsluta

B:
  Läs 8192 bytes från P
  Behandla de 8192 lästa byten på något sätt
  Skriv ett block med 4096 bytes till P
  Avsluta
```

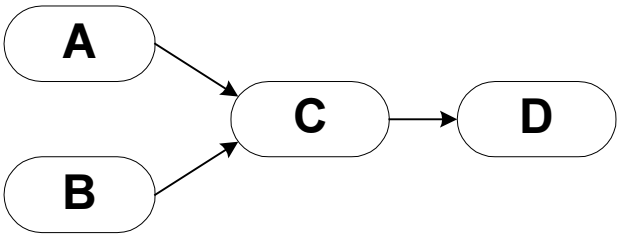
b) Syftet med nedanstående pseudokod är att överföra 1024 bytes från en föräldraprocess till dess barn.

```
Skapa en pipa P
Skriv 1024 bytes till P
Anropa fork() för att skapa ett barn
Om vi är föräldern:
        Stäng läsändan på P
        Stäng skrivändan på P
        Vänta på att barnet dör
Om vi är barnet:
        Stäng skrivändan på P
        Läs data från P, till EOF
        Stäng läsändan på P
        Behandla den lästa datan
        Avsluta
Avsluta
```

Programmet fungerar på vissa system, men inte på en del andra. Hur yttrar sig felet? Vad beror det på?

Uppgift 9 (6 p)

I en mättillämpning har man två processer, A och B, som avläser varsin temperaturgivare. Med oregelbundna mellanrum, när temperaturen förändrats tillräckligt mycket, skickar de nya mätvärden till varsin pipa. En tredje process, C, tar emot data från dessa båda pipor och skickar dem vidare på en pipa till en fjärde process, D:



Din uppgift är nu att skriva funktionen `transfer()`, som används i processen C för att läsa data från A och B och skicka dem vidare till D. Argumenten till funktionen är tre fildeskriptorer som är kopplade till lämpliga pipor/pipändar för att läsa från A och B respektive skriva till C:

```
void transfer(int fda, int fdb, int fdd);
```

Eftersom du inte vet när det finns data att läsa på `fda` och `fdb` måste du multiplexera dataöverföringen. Studera den `man`-sida för `poll()` som medföljer som bilaga till tentamen.

För att underlätta uppgiften kan du förutsätta följande:

- Processen körs i all evighet, dvs A och B kommer inte att stänga skrivändarna på sina pipor, och D kommer inte att stänga sin läsände.
- Varje mätvärde består av exakt 10 byte. Om `poll()` säger att det finns data att läsa på en deskriptor går det att läsa minst 10 byte från den deskriptorn.
- `poll()` misslyckas aldrig (dvs returnerar inte -1).
- Skrivning till D blockerar aldrig och behöver alltså inte multiplexeras.

Uppgift 10 (3+1=4 p)

a) För varje rad i programmet nedan som påverkar innehållet i `b[]`, visa hur innehållet i `b[]` ser ut efter det att raden exekverats.

b) Vad skrivs ut av programmet?

```
1   #include <stdio.h>
2   #include <string.h>

3   int main(void)
4   {
5      char b[25] = "Adrians granar",
6         *s[] = {
7            "esters astrar",
8            "lars-eriks och pers tall",
9            "sunes asp",
10           "veras al"
11        }, *r, **p, *a[3] = {*s, *(s+1), *(s+2)+5};

12     p = &a[1];
13     r = (char *) ((int *) b + 16/sizeof(int));
14     strcpy(--r, s[3]);
15     strcpy(b+2, *a+6);
16     strncpy(r=b+8, *p+14, 8);
17     b[19] = *(*s+strlen(*a));
18     strncpy(r+4, *(a+2), 4);
19     puts(b);

20     return 0;
21  }
```

NAME

poll - input/output multiplexing

SYNOPSIS

```
#include <poll.h>

int poll(struct pollfd fds[], nfds_t nfds,
     int timeout);
```

DESCRIPTION

The poll() function provides applications with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by **fds**, poll() examines the given file descriptor for the event(s) specified in **events**. The number of pollfd structures in the **fds** array is specified by **nfds**. The poll() function identifies those file descriptors on which an application can read or write data, or on which certain events have occurred.

The **fds** argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are pollfd structures, which contain the following members:

```
int fd;          /* file descriptor */
short events;    /* requested events */
short revents;   /* returned events */
```

The fd member specifies an open file descriptor and the events and revents members are bitmasks constructed by a logical OR operation of any combination of the following event flags:

POLLIN	Data other than high priority data may be read without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.
POLLRDNORM	Normal data (priority band equals 0) may be read without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.
POLLRDBAND	Data from a non-zero priority band may be read without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.
POLLPRI	High priority data may be received without blocking. For STREAMS, this flag is set in revents even if the message is of zero length.
POLLOUT	Normal data (priority band equals 0) may be written without blocking.
POLLWRNORM	The same as POLLOUT.
POLLWRBAND	Priority data (priority band > 0) may be written. This event only examines bands that have been written to at least once.
POLLERR	An error has occurred on the device or stream. This flag is only valid in the revents bitmask; it is not used in the events member.

POLLHUP — A hangup has occurred on the stream. This event and POLLOUT are mutually exclusive; a stream can never be writable if a hangup has occurred. However, this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not mutually exclusive. This flag is only valid in the revents bitmask; it is not used in the events member.

POLLNVAL — The specified fd value does not belong to an open file. This flag is only valid in the revents member; it is not used in the events member.

If the value fd is less than zero, events is ignored and revents is set to 0 in that entry on return from poll().

The results of the poll() query are stored in the revents member in the pollfd structure. Bits are set in the revents bitmask to indicate which of the requested events are true. If none are true, none of the specified bits are set in revents when the poll() call returns. The event flags POLLHUP, POLLERR, and POLLNVAL are always set in revents if the conditions they indicate are true; this occurs even though these flags were not present in events.

If none of the defined events have occurred on any selected file descriptor, poll() waits at least **timeout** milliseconds for an event to occur on any of the selected file descriptors. On a computer where millisecond timing accuracy is not available, **timeout** is rounded up to the nearest legal value available on that system. If the value **timeout** is 0, poll() returns immediately. If the value of **timeout** is INFTIM (or -1), poll() blocks until a requested event occurs or until the call is interrupted. poll() is not affected by the O_NDELAY and O_NONBLOCK flags.

The poll() function supports regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs and pipes. The behaviour of poll() on elements of **fds** that refer to other types of file is unspecified.

The poll() function supports sockets.

A file descriptor for a socket that is listening for connections will indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously will indicate that it is ready for writing, once a connection has been established.

Regular files always poll TRUE for reading and writing.

RETURN VALUES

Upon successful completion, a non-negative value is returned. A positive value indicates the total number of file descriptors that has been selected (that is, file descriptors for which the revents member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, a value of -1 is returned and errno is set to indicate the error.