UMEÅ UNIVERSITET
Institutionen för datavetenskap
Ola Ågren

Lösningsförslag Datavetenskap

Kurs:	**Systemprogrammering, 5p**
Tentamensdatum:	**28 Oktober 1997**
Hjälpmedel:	**Miniräknare**
Examinator:	**Ola Ågren (ola@cs.umu.se)**

G/3 = 30, 4 = 34.8, VG = 38, 5 = 39.6

Uppgift 1 (14p)

Skriv namn, personnummer, uppgift och sidnummer på därför avsedd plats på varje sida och använd endast ena sidan av varje papper.

Svar:

Inte allt för svårt hoppas jag…

Uppgift 2 (4p)

Förklara hur signaler fungerar och ge exempel på när man kan använda dessa.

Svar:

En signal är ett sätt att meddela att något speciellt har hänt, t ex när en användare vill avbryta ett program eller när man försöker dela med 0. Egentligen handlar det om att operativsystemet knackar en på axeln och lämnar över en siffra och beroende på vad som är inställt för just den signalen så kommer en av tre saker att hända: 1) Programmet kommer att avslutas (eventuellt kommer även en minneskopia att skapas, en sk core-dump), 2) Signalen ignoreras eller 3) En viss rutin anropas.

En signal kan t ex användas för att synkronisera två processer med varandra eller för att meddela att något har hänt.

Uppgift 3 (1+3=4p)

Rita och beskriv följande:

1. Aktiveringspost
2. Virtuella minnesutrymmet för en process

Svar:

1. Aktiveringspost; Ordningen kan vara annorlunda, men följande brukar finnas med:

Plats för parametrar	Återfinns ofta i stacken hos anropande funktion
Lokala variabler	
Sparade register	Om funktionen behöver ett register utan att ändra värdet
Föregående aktiveringspost	En pekare till denna kan finnas med
Återhoppsadress	Vart processen skall hoppa när den är färdig

2. Virtuella minnesutrymmet för en process:

Stack	Växer nedåt allt efter behov
Tomrum	Just nu ledigt minne
Allokerat minne/Heap	Program och systemstyrt, växer uppåt
Data	Här finns alla globala variabler
Text	Program och konstanta värden

Det finns maskiner där stacken och det allokerade minnet har bytt plats och sålunda växer åt andra hållet. Jag ger naturligtvis inte fel för ett sådant exempel.

Uppgift 4 (4p)

Nämn och beskriv fyra sätt att få processer att överföra större datamängder från den ena till den andra processen.

Svar:

pipor	Kan skicka data mellan processer med gemensam anfader
namngivna pipor (FIFOs)	En process kan öppna för läsning, en för skrivning
meddelandeköer	Man skickar meddelanden till en kö som någon annan process sedan kan hämta och läsa
delat minne	En process ändrar i minnet och alla andra ser det genast
minnesmappade filer	I princip samma sak som delat minne, men filen finns även på disk
filer	Data mellanlagras på disk innan den läses in i den andra processen
sockets	En process skapar en port på nuvarande maskin för andra processer att knyta upp sig och skicka meddelanden till

Uppgift 5 (6p)

Beskriv vad som händer när följande kommando exekveras.

```
% grep a file | wc > data
```

Hur startas processer och program? När, och av vilken process skapas pipor? Hur hanteras omdirigering etc? Beskriv utförligt och rita gärna figurer.

Svar:

Före:

csh (run)
0:term
1:term
2:term

Efter **pipe**:

csh (run)
0:term
1:term
2:term
3:pipe, read end
4:pipe, write end

Efter 2***fork**:

csh (run)
0:term
1:term
2:term
3:pipe, read end
4:pipe, write end

csh (run)
0:term
1:term
2:term
3:pipe, read end
4:pipe, write end

csh (run)
0:term
1:term
2:term
3:pipe, read end
4:pipe, write end

Efter 2***dup**:

csh (run)
0:term
1:term
2:term
3:pipe, read end
4:pipe, write end

csh (run)
0:term
1:pipe, write end
2:term
3:pipe, read end
4:pipe, write end

csh (run)
0:pipe, read end
1:term
2:term
3:pipe, read end
4:pipe, write end

Efter 6***close**:

csh (wait)
0:term
1:term
2:term

csh (run)
0:term
1:pipe, write end
2:term

csh (run)
0:pipe, read end
1:term
2:term

Efter **open**:

csh (wait)
0:term
1:term
2:term

csh (run)
0:term
1:pipe, write end
2:term

csh (run)
0:pipe, read end
1:data
2:term

Efter 2***exec**:

csh (wait)
0:term
1:term
2:term

grep (run)
0:term
1:pipe, write end
2:term

wc (run)
0:pipe, read end
1:data
2:term

Uppgift 6 (2p)

Vad är en inod och vad innehåller den. Beskriv någorlunda utförligt!

Svar:

En i-nod är det sätt som systemet håller reda på en fil. I i-noden finns information om hur många datablock en fil innehåller och var på den fysiska disken man kan återfinna innehållet i filen. Dessutom återfinns all annan information som man får med stat, t ex ägare och när den senast blev läst eller skriven.

Varje i-nod refereras med ett unikt heltal.

Uppgift 7 (6p)

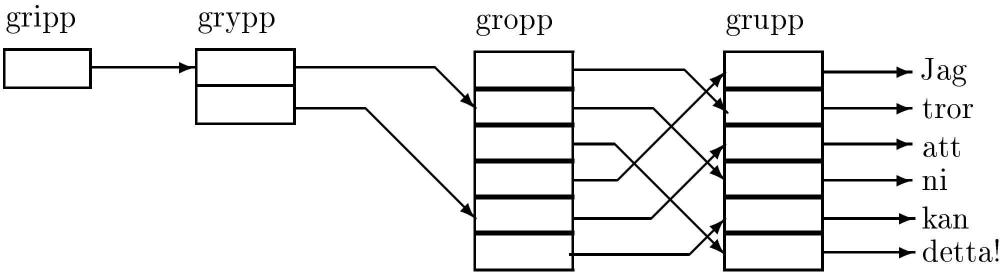
Rita hur pekarstrukturen ser ut vid exekvering och skriv dessutom vad som skrivs ut:

```
#include <stdio.h>
typedef char grepp[10];

grepp grupp[] = {"Jag", "tror", "att", "ni", "kan", "detta!"};
grepp *gropp[] =
    {&grupp[1], &grupp[3], &grupp[5], &grupp[0], &grupp[2], &grupp[4]};
grepp **grypp[] = {&gropp[1], &gropp[4]};
grepp ***gripp = grypp;

int main()
{
  printf("%c%s%c ", *(**gropp + 1), ***gripp + 1, *grupp[5]);
  putchar(*(***grypp + 1));
  printf(" %s%s%c\n", **(grypp[1] + 1) + 2, grupp[2], (*grypp[1][-2])[5]);
  exit(0);
}
```

Svar:




```
rid i natt!
```

Uppgift 8 (6p)

Ni har just fått anställning på aktiebolaget Minimal Softwares (**MS**). Eftersom det finns ett fel i programmet xargs på en av maskinerna så blir er första uppgift att skriva programmet yargs som skall fungera som xargs men som alltid tar en rad i taget från stdin och lägger detta sist i den kommandorad som sedan skall exekveras. Eftersom ni just nu inte har någon bok med er följer (efter första raderna ur xargs mansida) några funktionsprototyper som ni eventuellt kan ha använding av.

...

Svar:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char buf[1024];
    int i;

    for (i = 1; i < argc; i++)
        sprintf(buf, "%s%s ", buf, argv[i]);
    i = strlen(buf);

    while (scanf("%s", &buf[i]) > 0)
        system(buf);
    exit(0);
}
```

eller kanske

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

int main(int argc, char *argv[])
{
    char buf[1024];
    char **argv2;
    int i, pid;

    if ((argv2 = malloc((argc + 1) * sizeof(char*))) == NULL) {
        perror("malloc");
        exit(1);
    }
    for (i = 1; i < argc; i++)
        argv2[i - 1] = argv[i];
    argv2[argc - 1] = buf;
    argv2[argc] = NULL;

    while (scanf("%s", buf) > 0) {
        if ((pid = fork()) == 0) {
            execvp(argv2[0], argv2);
            perror(argv2[0]);
            exit(2);
        } else {
            while (pid != wait(NULL));
        }
    }
    free(argv2);
    exit(0);
}
```