UMEÅ UNIVERSITET
Institutionen för datavetenskap
Ola Ågren

Tentamen Datavetenskap

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

Födelsedatum: ____________________

Namn: ____________________

Namnteckning: ____________________

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

Markera de uppgifer som du har löst.

Uppgift	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Totalt**	**Bonus**
Löst										
Maxpoäng	**14**	**4**	**4**	**4**	**6**	**2**	**6**	**6**	**46**	**3.2**
Poäng										

Resultat/Betyg: ____________

Signatur: ____________

Uppgift 1 (14p)

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

Uppgift 2 (4p)

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

Uppgift 3 (1+3=4p)

Rita och beskriv följande:

1. Aktiveringspost
2. Virtuella minnesutrymmet för en process

Uppgift 4 (4p)

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

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.

Uppgift 6 (2p)

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

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);
}
```

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.

Exempel på exekvering: `ls | yargs wc`, kör wc på varje fil i biblioteket även när det finns för många filer för att direkt skriva `wc *`.

```
XARGS(1)                                                         XARGS(1)

NAME
     xargs - construct argument list(s) and execute command

SYNOPSIS
     xargs [flags] command [initial-arguments]

DESCRIPTION
     xargs combines the fixed initial-arguments with arguments read from
     standard input to execute the specified command one or more times.  The
     number of arguments read for each command invocation and the manner in
     which they are combined are determined by the flags specified.
```

Funktionsprototyper

int **execl**(const char **pathname*, const char **arg0*, .../* (char *) 0 */);
<unistd.h>
Returns: -1 on error, no return on success

int **execle**(const char **pathname*, const char **arg0*, .../* (char *) 0 , char *const *envp*[] */);
<unistd.h>
Returns: -1 on error, no return on success

int **execlp**(const char **filename*, const char **arg0*, .../* (char *) 0 */);
<unistd.h>
Returns: -1 on error, no return on success

int **execv**(const char **pathname*, char *const *argv*[]);
<unistd.h>
Returns: -1 on error, no return on success

int **execve**(const char **pathname*, char *const *argv*[], char *const *envp*[]);
<unistd.h>
Returns: -1 on error, no return on success

int **execvp**(const char **filename*, char *const *argv*[]);
<unistd.h>
Returns: -1 on error, no return on success

void **exit**(int *status*);
<stdlib.h>
This function never returns

int **fclose**(FILE **fp*);
<stdio.h>
Returns: 0 if OK, -1 on error

int **feof**(FILE **fp*);
<stdio.h>
Returns: nonzero (true) if end of file on stream, 0 (false) otherwise

int **fgets**(char **buf*, int *n*, FILE **fp*);
<stdio.h>
Returns: buf if OK, `NULL` on end of file or error

FILE ***fopen**(const char **filename*, const char **type*);
<stdio.h>
type: "r", "w", "a", "r+", "w+", "a+"
Returns: file pointer if OK, `NULL` on error

int **fprintf**(FILE **fp*, const char **format*, ...);
<stdio.h>
Returns: #characters output if OK, negative value if output error

void **free**(void **ptr*);
<stdlib.h>

int **fscanf**(FILE **fp*, const char **format*, ...);
<stdio.h>
Returns: #input items assigned, `EOF` if input error or `EOF` before any conversion

void ***malloc**(size_t *size*);
<stdlib.h>
Returns: nonnull pointer if OK, `NULL` on error

int **pclose**(FILE **fp*);
<stdlib.h>

Returns: 0 if OK, -1 on error

int **pipe**(int *fildes[2]*);
`<unistd.h>`
Returns: 0 if OK, -1 on error

FILE ***popen**(const char **cmdstring*, const char **type*);
`<stdlib.h>`
type: `"r", "w"`
Returns: file pointer if OK, `NULL` on error

int **printf**(const char **format*, ...);
`<stdio.h>`
Returns: #characters output if OK, negative value if output error

int **scanf**(const char **format*, ...);
`<stdio.h>`
Returns: #input items assigned, `EOF` if input error or `EOF` before any conversion

char ***strcpy**(char **s1*, const char **s2*);
`<string.h>`
Returns: s1 if OK, `NULL` on error

char ***strcat**(char **s1*, const char **s2*);
`<string.h>`
Returns: s1 if OK, `NULL` on error

int **system**(const char **cmdstring*);
`<stdlib.h>`
Returns: termination status of shell

pid_t **wait**(int **statloc*);
`<sys/types.h>`
`<sys/wait.h>`
Returns: process ID if OK, 0, or -1 on error

pid_t **waitpid**(pid_t *pid*, int **statloc*, int *options*);
`<sys/types.h>`
`<sys/wait.h>`
options: 0, WNOHANG, WUNTRACED
Returns: process ID if OK, 0, or -1 on error

Precedens och associativitet

<table>
<tr><th>Operator</th><th>Associativitet</th></tr>
<tr><td>() [] . -> ++ (postfix) -- (postfix)</td><td>vänster</td></tr>
<tr><td>++ (prefix) -- (prefix) ! ~ sizeof (typ)
+ (unart) - (unart) & (adress) * (dereferens)</td><td>höger</td></tr>
<tr><td>* / %</td><td>vänster</td></tr>
<tr><td>+ -</td><td>vänster</td></tr>
<tr><td><< >></td><td>vänster</td></tr>
<tr><td>< <= > >=</td><td>vänster</td></tr>
<tr><td>== !=</td><td>vänster</td></tr>
<tr><td>&</td><td>vänster</td></tr>
<tr><td>^</td><td>vänster</td></tr>
<tr><td>|</td><td>vänster</td></tr>
<tr><td>&&</td><td>vänster</td></tr>
<tr><td>||</td><td>vänster</td></tr>
<tr><td>?:</td><td>höger</td></tr>
<tr><td>= += -= *= /= %= >>= <<= &= ^= |=</td><td>höger</td></tr>
<tr><td>, (kommaoperator)</td><td>vänster</td></tr>
</table>