UMEÅ UNIVERSITET
Institutionen för datavetenskap
Ola Ågren

Tentamen Datavetenskap

Kurs:	**Systemprogrammering, 5p**
Tentamensdatum:	**19 November 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 (max en per papper) och sidnummer på därför avsedd plats på varje sida och använd endast ena sidan av varje papper.

Uppgift 2 (4p)

Det finns fem sätt att avsluta en process, beskriv dessa (en poäng per sätt, men inte fler än fyra räknas).

Uppgift 3 (4p)

Inom systemprogrammering pratar man ofta om processer och hur de interagerar med varandra. Definiera alt. föklara vad en process är samt rita och beskriv det virtuella minnesutrymmet för en process.

Uppgift 4 (4p)

Nämn och beskriv fyra sätt att få processer att synkroniseras med varandra.

Uppgift 5 (6p)

Beskriv vad som händer när följande kommando exekveras.

```
% sed -e script < indata | tee utdata
```

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 hård länk? Vad är en symbolisk länk? Beskriv någorlunda utförligt likheter och skillnader dem emellan!

Uppgift 7 (6p)

Rita hur pekarstrukturen ser ut vid början av exekveringen 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()
{
  char *grapp = (**gropp) + 2;

  (*grypp[1][1])[2] = '\0';
  (*grypp[1][-2])[5] = '\0';
  printf("%s%s", grapp++, grypp[1][1]);
  printf("%s %s", grapp, ***gripp);
  printf(" %s?\n", **(gropp + 2));
  exit(0);
}
```

Uppgift 8 (6p)

Ni har just fått anställning på aktiebolaget Minimal Softwares (**MS**). Eftersom det finns ett fel i programmet `wc` på en av maskinerna så blir er första uppgift att skriva programmet `mwc` som skall fungera som `wc -cl`. Eftersom ni just nu inte har någon bok med er följer (efter mansidan för `wc`) några funktionsprototyper som ni eventuellt kan ha använding av.

```
WC(1)                                                                WC(1)

NAME
     wc - word count

SYNOPSIS
     wc [ -lwc ] [ names ]

DESCRIPTION
     wc counts lines, words, and characters in the named files, or in the
     standard input if no names appear.  It also keeps a total count for all
     named files.  A word is a maximal string of characters delimited by
     spaces, tabs, or new-lines.

     The options l, w, and c may be used in any combination to specify that a
     subset of lines, words, and characters are to be reported.  The default
     is -lwc.

     When names are specified on the command line, they will be printed along
     with the counts.

                                                                    Page 1
```

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 **fgetc**(FILE *fp);
<stdio.h>
Returns: next character if OK, EOF on end of file or error

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

pid_t **fork**(void);
<sys/types.h>
<unistd.h>
Returns: 0 in child, process ID of child in parent, -1 on error

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

int **fputc**(int c, FILE *fp);
<stdio.h>
Returns: c if OK, EOF on error

int	**fputs**(const char **str*, FILE **fp*);	
		`<stdio.h>`
		Returns: nonnegative value if OK, `EOF` on 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
int	**getc**(FILE **fp*);	
		`<stdio.h>`
		Returns: next character if OK, `EOF` on end of file or error
int	**getchar**(void);	
		`<stdio.h>`
		Returns: next character if OK, `EOF` on end of file or error
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
void	**perror**(const char **msg*);	
		`<stdio.h>`
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	**putc**(int *c*, FILE **fp*);	
		`<stdio.h>`
		Returns: *c* if OK, `EOF` on error
int	**putchar**(int *c*);	
		`<stdio.h>`
		Returns: *c* if OK, `EOF` on error
int	**puts**(const char **str*);	
		`<stdio.h>`
		Returns: nonnegative value if OK, `EOF` on 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>