UMEÅ UNIVERSITET
Institutionen för datavetenskap
Gunnar Gunnarsson, Tomas Halvarsson

Tentamen Datavetenskap

Kurs: Systemprogrammering, 5p
Tentamensdatum: 27 oktober 1998 kl 9-15
Hjälpmedel: *A book on C* / *Vägen till C*
Examinator: Gunnar Gunnarsson (gunnar@cs.umu.se)

Personnummer: _____

Namn: _____

Namnteckning: _____

Maxpoäng: 40 poäng
Betygsgränser: 20 poäng (betyg 3), 26 poäng (betyg 4), 32 poäng (betyg 5)

Kryssa för de uppgifter som du har löst.

Uppgift	1	2	3	4	5	6	7	8	9	Totalt	Bonus
Löst											
Maxpoäng	4	4	4	6	2	2	6	6	6	40	2 (4)
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) shellskript
b) systemanropsfunktion
c) demonprocess
d) zombie
e) set-user-ID-program
f) kernel
g) race condition
h) core dump

Uppgift 2 (1+1+1+1=4 p)

a) Hur deklarerar man i C en array (A) med fem element av typen pekare till int?

b) Hur deklarerar man i C en pekare (P) på en array med fem element av typen int?

c) Beskriv med ord vilken typ X har i följande deklaration:

```
int **X[5];
```

d) Beskriv med ord vilken typ Y har i följande deklaration:

```
int *(*Y)(int *, char **);
```

Uppgift 3 (3+1=4 p)

a) En interpreter-fil ser i princip ut på följande sätt:

```
#!/path/interpreter-program argument
rad 1
rad 2
...
rad N
```

Förklara vad som händer när `execvp()` anropas för att köra en sådan fil. Två av de saker du bör beskriva är när och av vilka processer interpreter-filen öppnas, samt hur kommandoraden till interpreter-programmet ser ut.

b) Vad är vitsen med interpreter-filer?

Uppgift 4 (2+2+1+1=6 p)

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

char *caps(const char *s)
{
    char *ns;
    int i;

    ns = malloc(strlen(s));
    if (ns = NULL) return NULL;

    for (i = 0; s[i] != '\0'; i++)
        ns[i] = ( (s[i] >= 'a' && s[i] <= 'z')
            ? (s[i] - ('a'-'A'))
            : s[i]);

    return ns;
}
```

Funktionen `caps()` ska ta en sträng och returnera en pekare till en ny sträng, som har samma innehåll förutom att alla gemener (a-z) i den gamla strängen ska vara versaler (A-Z) i den nya strängen. `caps()` ska själv allokera upp minne för den nya strängen. Om det inte går att allokera minne ska `caps()` returnera NULL.

Exempel: `caps("Sune")` ska returnera en pekare till strängen `"SUNE"`.

Koden kommer tyvärr inte att fungera som avsett, på grund av att den innehåller två mycket allvarliga fel.

a) Vilka är felen, och hur borde det vara istället?

b) Hur tror du att felen kommer att yttra sig när funktionen körs på de Sun Solaris-system som använts på laborationerna? Beskriv felen vart och ett för sig, dvs när du förklarar hur det ena felet yttrar sig ska du förutsätta att det andra felet är åtgärdat. Glöm inte att motivera dina teorier.

c) Varför är det så viktigt att testa returvärdet från `malloc()`, `open()` och liknande systemanropsfunktioner?

d) Varför är det så viktigt att skriva ut eventuella felmeddelanden på stderr och inte på stdout?

Uppgift 5 (1+1=2 p)

Som du kanske minns från föreläsningarna är en av nyheterna i version 4.0 av Windows NT att stora delar av grafikhanteringen har flyttats från user mode till kernel mode, huvudsakligen därför att det ger bättre prestanda.

a) Vad är det för skillnad på kernel mode och user mode?

b) Varför kör man inte hela operativsystemet i kernel mode?

Uppgift 6 (1+1=2 p)

a) Om signalen SIGINT levereras till en process beror det på att användaren har tryckt ^C (Control-C). Ge ytterligare två exempel på saker som kan leda till att en process får en signal.

b) En process som får signalen SIGINT avbryts normalt. Det finns flera olika metoder för processen att förhindra detta. Förklara kortfattat en av dessa metoder.

Uppgift 7 (6 p)

Skriv ett interpreter-program för språket VSPL (Very Simple Programming Language). Språket har två nyckelord:

print <sträng> skriver ut <sträng> på stdout.

run <program> startar det externa programmet <program> och väntar tills <program> avslutats. Körningen av interpreter-filen (VSPL-programmet) fortsätter sedan. Program som startas med run kan inte få några kommandoradsargument. Pipor och omdirigering behöver du inte heller bekymra dig om.

Nyckelorden står alltid först på en rad. Om interpreter-programmet stöter på ett ogiltigt nyckelord ska programmet avslutas med ett felmeddelande.

Exempel på interpreter-fil:

```
#!/Home/staff/gunnar/vspl
print *** Ett enkelt testprogram ***
print Följande filer finns i aktuell katalog:
run ls
print Följande användare är inloggade:
run who
print Körningen avslutas nu.
```

Uppgift 8 (6 p)

Skriv en funktion som tar en sträng som innehåller engelsk text, och returnerar antalet felstavade ord i texten. Funktionshuvud:

```
int cnt_err(const char *s)
```

Tips: Kommandot ”ispell -l” läser data från stdin. Vid filslut skriver ispell ut en lista med alla felstavade ord. Exempel:

```
peppar% ispell -l
Say hello to my litle frend.
^D
litle
frend
peppar%
```

`^D` i exemplet ovan betyder att användaren tryckt Control-D för att ange filslut, och `peppar%` är förstås shell-prompten. Det som står i fet stil är det användaren skrivit in på tangentbordet.

Du kan ha nytta av pipor för att lösa uppgiften. Försök däremot inte använda `popen()` eller `system()`. Lösningar som bygger på temporärfiler godkänns inte.

Uppgift 9 (1+1+2+2=6 p)

a) Vilket innehåll har arrayen `a[]` sedan nedanstående kod exekverats?

```
char a[5] = ”ABCD”, *p = a;
*(p+3) = ’E’;
```

b) Vad är skillnaden på innehållet i `a` och innehållet i `b` efter följande deklaration?

```
char a[] = ”ABCD”,
     b[] = {’A’, ’B’, ’C’, ’D’};
```

c) Visa för varje rad i programmet på nästa sida hur innehållet i `d[]` ser ut efter det att raden exekverats. Du behöver inte förklara rader som inte påverkar `d[]`.

d) Vad skrivs ut av programmet på nästa sida? Tips: Titta noga på vad som görs i for-slingan, annars blir det fel.

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

int main(void)
{
    int i;
    char **p, *a[] = {
        "ATroaat####",
        "slirp%slurp",
        "atei!!!!",
        "Gluff gluff"
    }, d[20] = "Tipp tapp", *s;

    p = a;
    strncpy(d+1, &a[1][1], 7);
    *(d+1) = *p[3];
    s = &p[3][10];
    strcpy(&d[7], s);
    s = strcat(d, p[2]);
    *(d+5) = d[8];
    d[8] = a[2][2];
    d[7] = d[10] = p[0][6];
    d[13] = *(s+6);

    for (i = 0; i < 16; i++)
        printf("%c", d[++i]);
    printf("\n");

    return 0;
}
```

Funktionshuvudet för strncpy finns inte i bilagan. Det är:

```
char *strncpy(char *dst, const char *src, size_t n);
```

Funktionen ligger i <string.h> och returnerar `dst`.