Fachhochschule Nordwestschweiz
Hochschule für Technik

System-Programmierung (syspr) 5. November 2019 thomas.amberg@fhnw.ch

Assessment I

Vorname: Mike

Name: Nöthiger

Klasse: 3ia

Punkte: 85 / 92, Note: 5.67

Frei lassen für Korrektur.

Hilfsmittel:

- Die vom Dozent ausgeteilte C Referenzkarte.
- Lösen Sie die Aufgaben direkt auf den Prüfungsblättern.
- Zusatzblätter, falls nötig, mit Ihrem Namen und Fragen-Nr. auf jedem Blatt.

Nicht erlaubt:

- Unterlagen (Slides, Bücher, ...).
- Computer (Laptop, Smartphone, ...).
- Kommunikation mit anderen Personen.

Bewertung:

- Multiple Response: ☐ *Ja* oder ☐ *Nein* ankreuzen, `+1/-1` Punkt pro richtige/falsche Antwort, beide nicht ankreuzen ergibt `+0` Punkte; Total pro Frage gibt es nie weniger als `0` Punkte.
- Offene Fragen: Bewertet wird Korrektheit, Vollständigkeit und Kürze der Antwort.
- Programme: Bewertet wird die Skizze/Idee und Umsetzung des Programms.

Fragen zur Prüfung:

- Während der Prüfung werden vom Dozent keine Fragen zur Prüfung beantwortet.
- Ist etwas unklar, machen Sie eine Annahme und notieren Sie diese auf der Prüfung.

Erste Schritte in C

1) Welche dieser Ausdrücke kompilieren fehlerfrei in C? Punkte: 4 / 4

Zutreffendes ankreuzen; C = C99:

- ☐ Ja | ☒ Nein `int i = malloc(sizeof(int));`
- ☒ Ja | ☐ Nein `char s[] = {'o', 'o', 'p', 's'};`
- ☐ Ja | ☒ Nein `byte *b = "hello";`
- ☒ Ja | ☐ Nein `int j = (0 || 1);`

2) Was sind drei Unterschiede der Variablen x und y im folgenden Code? Punkte: 6 / 6

```
void g(int *p) { … }

void f() {
    int *x = malloc(sizeof(int));
    g(x);

    int y;
    g(&y);
}
```

Unterschiede hier eintragen, jeweils beide Seiten des Unterschieds ausformulieren:

y ist auf Stack alloziert	x ist auf Heap alloziert
x ist ein Pointer	y ist ein primitiver Datentyp (int)
Wert von x ist die Adresse, auf die x zeigt (✓)	Wert von y ist der int Wert, der hinterlegt ist an der Adresse von y (✓)

Funktionen in C

3) Schreiben Sie ein Programm, das seine Argumente rückwärts ausgibt: Punkte: 12 / 12

```
$ ./my_program just a test
tset a tsuj
```

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
int printf(const char *format, ...); // format string %s, char %c, int %d
size_t strlen(const char *s); // calculate the length of a string
```

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

Idee: 1. Rückwärts über Programm args iterieren ab argv[1]
2. Rückwärts über char* iterieren
3. printen

```
int main(int argc, char *argv[]) {
    for(int i= argc-1; i>=1; i--) {
        size_t n = strlen(argv[i]);
        for(int j= n-1; j>=0; j--) {
            printf("%s", argv[i][j]);
        }
        printf(" ");
    }
    printf("\n");
}
```

Da ich mir nicht sicher bin, ob das geht, eine Alternative Form wäre:
*(argv[i]+j)

4) Gegeben den folgenden Code, welchen Wert hat *k* nach Aufruf von *f()*? Punkte: 3 / 4

```
int f(int *m, int n) {
   return (*m + 1) * n;
}

int main() {
    int i[] = {3, 5};
    int j = 2;
    int k = f(i, j);
}
```

dereferenziert erstes Element in array

ist automatisch Pointer auf 1. Element im array

i <=> &i[0]

Resultat und Begründung hier eintragen:

=> 4·2 => k = 8

Begründung im Code.

5) Gegeben den folgenden Code, welche Aufrufe von *eval()* sind erlaubt? Punkte: 4 / 4

```
int inc(int i) { return i++; }
int add(int a, int b) { return a + b; }
int eval(int a, int b, int (*op)(int, int)) { return op(a, b); }
```

Zutreffendes ankreuzen:

- ☒ Ja | ☐ Nein `eval(inc(3), 3, add);`
- ☐ Ja | ☒ Nein `eval(add(3, 3), 3, inc);`
- ☒ Ja | ☐ Nein `eval(add(3, 3), inc(3), add);`
- ☒ Ja | ☐ Nein `eval(eval(3, 3, add), 3, add);`

6) Gegeben die Listen-Datenstruktur (unten), implementieren Sie *insert()* um für den String *item* einen Node zu allozieren und alphabetisch sortiert in die Liste einzufügen. Punkte: 12 / 12

```
typedef struct node {
    struct node *next;
    char *item;
} Node;

static Node *list = NULL;

void insert(char *item); // TODO: implement
```

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

`void *malloc(size_t size); // Allocates size bytes and returns a pointer to the allocated memory.`
`int strcmp(char *s1, char *s2); // Compares strings s1, s2 alphabetically and returns an integer less than (s1<s2), equal to, or greater than zero.`

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

Idee: 1. Speicher für Node allozieren, item Wert setzen

2. wenn list == NULL, list = n;

3. prev finden:

```
prev = NULL;
pivot = list;
while (strcmp(n->item, pivot->item) > 0 && pivot->next
    pivot = pivot->next;
    prev = pivot;
```

4. if (prev == NULL): n->next = list;
   list = n;

else if (prev->next == NULL): prev->next = n;

else: n->next = prev->next;
   prev->next = n;

einfügen (✓)

```
node *create(char *item) {
    Node *n = malloc(sizeof(Node));
    n->next = NULL;
    n->item = item;
    return n;
}

int main() {

void insert(char *item) {
    Node *n = create(item);
    if(list == NULL) {
        list = n;
        return;
    }
    Node *prev = NULL;
    Node *pivot = list;
    while(strcmp(n->item, pivot->item) > 0 && pivot->next != NULL){
        pivot = pivot->next;
        prev = pivot;
    }
    if (prev == NULL) { n->next = list; list = n; return; }
    else if( prev->next == NULL) { prev->next = n; return; }
    else {
        n->next = prev->next;
        prev->next = n;
        return;
    }
}
```

File In-/Output

7) Gegeben eine ASCII Datei im *Comma Separated Value (CSV)* Format, z.B.: Punkte: 14/ 16

```
$ cat table.csv
Animal, Size\n
Dog, Big\n
Duck, Small\n
Mouse, Tiny\n
Horse, Huge\n
```

Schreiben Sie ein Programm, das eine Spalte der Form *Titel Wert* ... an die Tabelle anhängt.

Annahme 1: Der Name der Output Datei wird als Command Line Parameter übergeben.

Annahme 2: Die Anzahl Elemente in der Spalte passt zur Anzahl Zeilen in der Tabelle.

```
$ ./my_program table.csv table2.csv Legs 4 2 4 4

$ cat table2.csv
Animal, Size, Legs\n
Dog, Big, 4\n
Duck, Small, 2\n
Mouse, Tiny, 4\n
Horse, Huge, 4\n
```

Nutzen Sie dazu diese System-Calls, #includes und Fehlerbehandlung können Sie weglassen:

```
int close(int fd); // Closes the file descriptor fd.
```

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode); // Opens the file
specified by pathname. Or creates it if O_CREAT is used. Returns the file
descriptor. Flags include O_APPEND, O_CREAT, O_RDONLY, O_WRONLY, O_RDWR.
Modes, which are used together with O_CREAT include S_IRUSR and S_IWUSR.
```

```
ssize_t read(int fd, void *buf, size_t n); // Attempts to read up to n
bytes from file descriptor fd into buf. Returns number of bytes read ≤ n.
```

```
ssize_t write(int fd, const void *buf, size_t n); // Writes up to n bytes
from buf to the file referred to by fd. Returns nr. of bytes written ≤ n.
```

(Fortsetzung auf der nächsten Seite)

(7) Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen & Fragen-Nr.:

Idee: ~~1. open file O_RDWR~~ returns ~~=>~~ fd

~~2. create method that moves offset of fd before last \n of current line~~

~~3.1~~

1. open input O_RDONLY => fdi;
2. open output O_WDONLY => fdo;
3. create method void append_to_line (char *text) that behaves as follows:
   3.1 read sizeof(char) from fdi and store to a char c;
   3.2 As long as c != "\n" write c to fdo
   3.3 When c is "\n" write text to fdo, then write "\n" to fdo
4. Call append_to_line for argv[3] to argv[argc-1]
5. close fdi and fdo.

zu ausführlich

~~Code next page~~

Code on back of page 6

```
int main (int argc, char *argv[]) {
    int fdi = open(argv[1], O_RDONLY);
    int fdo = open(argv[2], O_WRONLY | O_CREATE);

    for(int i=3; i<argc; i++) {

        append_to_line(fdi, fdo, argv[i]);
    }
    *(1)
}

void append_to_line (int fdi, int fdo, char *text) {
    char *c; = malloc ... (oder einfach
                                  char c[1];)
    size_t n = sizeof(char);
    while (*c != '\n' && !EOF) {
        read(fdi, c, n);
        write(fdo, c, n);
    }
    write(fdo, text, sizeof(*text));
    write(" ", ")
    *c = '\n';
    write(fdo, *c, sizeof(*c));
}

*(1) close(fdi);
     close(fdo);
```

Prozesse und Signale

8) Schreiben Sie ein Programm, das einen Zombie-Prozess erzeugt, für 1 Sekunde. P.: 8 / 8

```
$ ./my_program &
[1] 1001
$ ps aux | grep my_program
... 1001 ... ./my_program
... 1002 ... [my_program] <defunct> // Zombie Prozess
```

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

void **exit**(int status); // Cause normal process termination.
pid_t **fork**(void); // Create a child process; returns 0 in child process.
int **sleep**(unsigned int seconds); // Causes the calling thread to sleep.
pid_t **wait**(int *wstatus); // Wait for child process; wstatus can be NULL.

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

Idee: Forken, Child Prozess direkt Enden lassen, Parent Prozess für 1 sek auf sleep stellen, danach child mit wait abholen.

```
int main() {
    pid_t pid = fork();
    if(pid == 0) exit(0);
    else {
        sleep(1);
        wait(NULL);
        exit(0);
    }
}
```

9) Schreiben Sie ein Programm das beweist, dass beim Beenden eines Child Prozesses jeweils ein SIGCHLD Signal an den Parent Prozess gesendet wird.

Punkte: 9 / 9

`void exit(int status); // Cause normal process termination.`
`pid_t fork(void); // Create a child process; returns 0 in child process.`
`int pause(void); // Pause causes the calling process to sleep until a signal terminates the process or causes invocation of a handler function.`
`int printf(const char *format, …); // format string %s, char %c, int %d`
`typedef void (*sighandler_t)(int);` `sighandler_t signal(int signum, sighandler_t handler); // set SIG_IGN, SIG_DFL, or a programmer-defined function to handle the signal signum.`

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

Idee: 1. Forken ✓
2. Child enden lassen ✓
3. Pausieren bis Signal
4. Signal Handler registrieren
5. Signal Printen

```
void handler(int signum) {
    printf("received %d\n", signum);
}
int main() {
    signal(SIGCHLD, handler);
    if(fork() == 0) exit(0);
    else {
        pause();
    }
    return 0;
}
```

Prozess-Lebenszyklus

10) Schreiben Sie ein Programm, dass sich selbst genau einmal erneut startet. Punkte: 9 / 9

```
$ ./my_program
main
main
```

Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
int execve(const char *pathname, char *const argv[], char *const envp[]);
// execute a program; argument vector and environment are passed to main
pid_t fork(void); // create a child process; returns 0 in child process
int printf(const char *format, …); // format string %s, char %c, int %d
pid_t wait(int *wstatus); // wait for child process; wstatus can be NULL
```

Idee (kurz) und Source Code hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

Idee: 1. Wenn programm argumente > 1, dann zum 2. Mal gestart

2. printf("main")
3. if zum 1. Mal gestartet
   fork() *
   child: execve(argv[0], args, NULL);
   exit(0)
   parent: wait(NULL)

Zu lang

Programm auf Rückseite →

* char *args[2];
args[0] = argv[0]
args[1] = "2";

```
int main(int argc, char *argv[]) {
    printf("main\n");
    if (argc <2) return 0;
    pid_t pid = fork();
    if( fork ==0) {
        char *args[2];
        args[0] = argv[0];
        args[1] = "2";
        execve(argv[0], args, NULL);
        exit(0);
    } else {
        wait(NULL);
    }
}
```

Threads und Synchronisation

11) Was ist der Output dieses Programms, und wieso?

Punkte: 4 / 8

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;

void *start(void *arg) {
    pthread_mutex_lock(&m);
    printf("A\n");
    pthread_cond_signal(&c);
    pthread_mutex_unlock(&m);
}

int main() {
    printf("B\n");
    pthread_t thread;
    pthread_mutex_lock(&m);
    pthread_create(&thread, NULL, start, NULL);
    printf("C\n");
    pthread_cond_wait(&c, &m);
    printf("D\n");
    pthread_mutex_unlock(&m);
}
```

Output und Begründung hier eintragen; Annahme: #includes sind vorhanden.

B
C

main thread takes mutex, thus start thread is blocked
main thread waits for condition, but does not free mutex befor. => Thus it's a deadlock, program won't proceed