

System-Programmierung (syspr)

03. Juni 2019

thomas.amberg@fhnw.ch

## Assessment II

Vorname:	Punkte: / 76, Note:
Name:	Frei lassen für Korrektur.
Klasse: □ Klasse 4ibb1 ⊠ Klasse 4ibb2	
Hilfsmittel:	
- Die vom Dozent ausgeteilte C Referenzkarte.	
- Lösen Sie die Aufgaben direkt auf den Prüfungsblätte	ern.
- Zusatzblätter, falls nötig, mit Ihrem Namen und Frag	gen-Nr. auf jedem Blatt.
Nicht erlaubt:	
- Unterlagen (Slides, Bücher,).	
- Computer (Laptop, Smartphone,).	
- Kommunikation mit anderen Personen.	
Bewertung:	
- Multiple Choice: Eine $\boxtimes$ <i>Antwort</i> pro Frage ankreuze	en, 4 Punkte pro richtige Antwort.
- Offene Fragen: Bewertet wird Korrektheit, Vollständi	igkeit und Kürze der Antwort.
- Programme: Bewertet wird die Skizze/Idee und Umse	etzung des Programms.
Fragen zur Prüfung:	
- Während der Prüfung werden vom Dozent keine Frag	gen zur Prüfung beantwortet.

- Ist etwas unklar, machen Sie eine Annahme und notieren Sie diese auf der Prüfung.

## Threads & Synchronisation

1) Beschreiben Sie die Funktionsweise und Mechanismen dieses Programms. Punkte: \_\_\_ / 6
Was simuliert das Programm, wie funktioniert es, und wie heisst das allgemeine Pattern?

```
#define OVEN_MAX_PIZZAS 3
01
02
03
    pthread_mutex_t oven_mutex = PTHREAD_MUTEX_INITIALIZER;
04
    void *baker_start(void *arg) {
05
        while (1) {
06
            pthread_mutex_lock(&oven_mutex);
07
            if (oven_n_pizzas() < OVEN_MAX_PIZZAS) {</pre>
98
                 int p = create_pizza();
09
                 oven_insert_pizza(p);
10
11
            pthread_mutex_unlock(&oven_mutex);
12
13
        }
14
    }
15
    void *waiter_start(void *arg) {
16
17
        while (1) {
            pthread_mutex_lock(&oven_mutex);
18
            if (oven_n_pizzas() > 0) {
19
                 int q = oven_remove_pizza();
20
                 serve_pizza(q);
21
22
23
            pthread_mutex_unlock(&oven_mutex);
24
        }
25
    }
26
    ... // implementations of application specific functions, oven_..., etc.
27
28
29
    int main () ... // calls baker_start, waiter_start in separate Threads
```

Freitext Antwort hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:

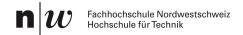
2) Schreiben Sie den folgenden Code um, so dass stack\_push() Thread-safe ist. Punkte: \_ / 6
Thread-safe heisst hier "ohne Race Conditions" falls mehrere Threads die Funktion aufrufen.

```
01
    typedef struct node {
02
        struct node *next;
03
        int item;
04
    } Node;
05
06
    static Node *stack = NULL;
07
98
    void stack_push(int item) {
        Node *n = malloc(sizeof(Node));
09
10
        n->item = item;
11
        n->next = stack;
        stack = n;
12
13
    }
14
    int stack_pop() { ... } // ignore
15
```

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

```
int pthread_mutex_lock(pthread_mutex_t *mutex); // lock a mutex
int pthread_mutex_unlock(pthread_mutex_t *mutex); // unlock a mutex
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // initialize a mutex

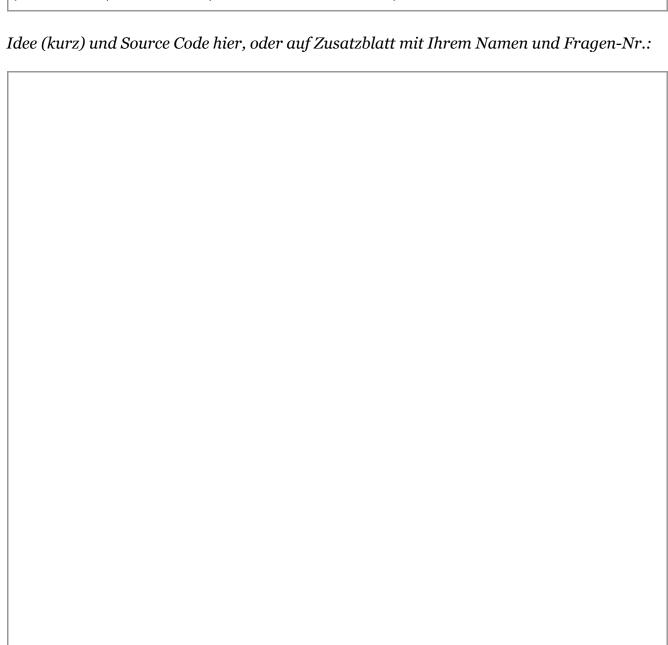
void *malloc(size_t size); // allocate dynamic memory; thread-safe
```



## **IPC** mit Pipes

3) Schreiben Sie ein Programm, das seine Child Prozesse per Pipe verbindet. Punkte: \_\_\_ / 12 Child Prozess Nr. 1 soll die Nachricht "hi" an Child Prozess Nr. 2 senden, über diese Pipe. Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
int close(int fd); // close a file descriptor
pid_t fork(void); // create a child process, returns 0 in child process
int pipe(int pipe_fd[2]); // create pipe, from pipe_fd[1] to pipe_fd[0]
ssize_t read(int fd, void *buf, size_t count); // read from a file descr.
ssize_t write(int fd, const void *buf, size_t count); // write to a file
pid_t wait(int *status); // wait for child process, status can be NULL
```



# Sockets

JNIX Domain Datagram Sockets	Internet Datagram Sockets (UDP)
Wenn Sie <u>http://fhnw.ch/</u> im Browser	r öffnen, wer ruft <i>read()</i> auf, und wozu? Punkte: _
size_t <b>read</b> (int fd, void *buf,	r öffnen, wer ruft read() auf, und wozu? Punkte: _ size_t count); // read bytes from socket latt mit Ihrem Namen und Fragen-Nr.:
size_t <b>read</b> (int fd, void *buf,	<pre>size_t count); // read bytes from socket</pre>
size_t <b>read</b> (int fd, void *buf,	<pre>size_t count); // read bytes from socket</pre>
size_t <b>read</b> (int fd, void *buf,	<pre>size_t count); // read bytes from socket</pre>
size_t <b>read</b> (int fd, void *buf,	<pre>size_t count); // read bytes from socket</pre>



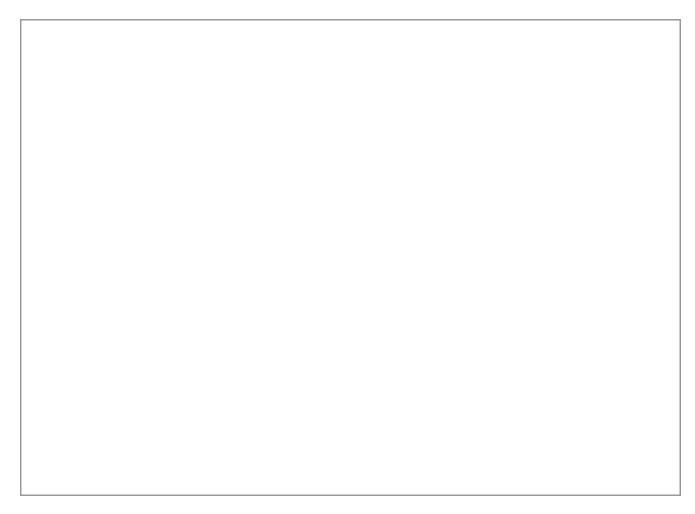
6) Welche Reihenfolge haben die folgenden Calls bei einem UDP Server? Punkte: \_ / 4

```
int bind(int sock_fd, const struct sockaddr *addr, socklen_t addrlen);
 ssize_t recvfrom(int fd, void *buf, size_t len, int flags,
   struct sockaddr *addr, socklen_t *addr_len);
 ssize_t sendto(int fd, const void *buf, size_t len, int flags,
   const struct sockaddr *dest_addr, socklen_t addrlen);
 int socket(int domain, int type, int protocol);
Eine Antwort ankreuzen:
 □ socket(); bind(); sendto(); recvfrom();
 □ bind(); socket(); sendto(); recvfrom();
 \square socket(); bind(); recvfrom(); sendto();
 □ bind(); sendto(); socket(); recvfrom();
 □ socket(); sendto(); bind(); recvfrom();
 □ bind(); socket(); recvfrom(); sendto();
Terminals
7) Welcher Terminal Mode wird für Text-Editoren verwendet, und wieso?
                                                                     Punkte: /3
Freitext Antwort hier, oder auf Zusatzblatt mit Ihrem Namen und Fragen-Nr.:
```



#### **POSIX IPC**

8) Schreiben Sie ein Programm, das den Verkauf von 99 Tickets durch 7 Verkäufer simuliert. Verkäufe (parallel) dauern 1-3s, ein Semaphor soll garantieren, dass es 99 sind. Punkte: \_ / 14 Hier ein Auszug aus der Doku, #includes und unerwartete Fehler können Sie weglassen:



### Zeitmessung

9) Schreiben Sie ein Programm, das misst, wie lange (Echtzeit) ein Signal hat. Punkte: \_\_\_/ 12
Die Messung soll beim Senden von *SIGUSR1* starten und direkt nach dem Eintreffen stoppen.

```
$ ./my_signal_time
Sending signal ...
... handling done.
real: 67999 10^-9s
```

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

```
int printf(const char *frmt, ...); // frmt int %d, double %lf, string %

typedef void (*sighandler_t)(int); // handler signature
    sighandler_t signal(int signum, sighandler_t handler); // install handler
    int raise(int sig); // send a signal to the caller, returns 0 or non-zero

int clock_gettime(clockid_t c_id, struct timespec *s); // returns 0 or -1

// c_id = CLOCK_REALTIME or CLOCK_MONOTONIC or CLOCK_PROCESS_CPUTIME_ID
    struct timespec {
        time_t tv_sec; // seconds
        long tv_nsec; // nanoseconds
    };
```

10) Schreiben Sie ein Programm, das "morgen", 00:00:00 als Datum ausgibt. Punkte: \_\_\_ / 9

Der Output des Programms soll so aussehen, mit dem aktuellen Datum, im Default-Format:

```
$ ./tomorrow
today is: Sun Jun 2 16:40:44 2019
tomorrow: Mon Jun 3 00:00:00 2019
```

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

```
int printf(const char *frmt, ...); // frmt int %d, double %lf, string %s
time_t time(time_t *t); // get local time in seconds since Epoch
char *ctime(const time_t *t); // convert t to ASCII, default date format
struct tm *localtime(const time_t *t); // get broken-down local time
time_t mktime(struct tm *tm); // convert broken-down local time to time_t
// ignores tm_wday, tm_yday; values outside valid interval are normalised
struct tm {
  int tm_sec; // Seconds (0-60)
  int tm_min; // Minutes (0-59)
  int tm_hour; // Hours (0-23)
  int tm_mday; // Day of the month (1-31)
  int tm_mon; // Month (0-11)
  int tm_year; // Year - 1900
  int tm_wday; // Day of the week (0-6, Sunday = 0)
  int tm_yday; // Day in the year (0-365, 1 Jan = 0)
  int tm_isdst; // Daylight saving time
};
```



Zusatzblatt zu Aufgabe Nr	von (Name)	