Fachhochschule Nordwestschweiz
Hochschule für Technik

System-Programmierung (syspr) 07. Januar 2019 thomas.amberg@fhnw.ch

Assessment II

Vorname: Mike

Name: Nöthiger

Klasse: 3ia

Punkte: 84 / 90, Note: 5.77

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.

Threads & Synchronisation

1) Programmieren Sie "sleep sort", ein Algorithmus, der Zahlen sortiert, indem er für jede Zahl *n* in einem eigenen Thread *sleep(n)* aufruft, und dann die Zahl ausgibt. Punkte: 15/ 16

```
$ ./my_sleep_sort 4 2 1 5
1 2 4 5
```

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

```
int atoi(const char *nptr); // convert a string to an integer
int printf(const char *format, ...); // format string %s, char %c, int %d
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
  void *(*start) (void *), void *arg); // starts a thread; attr = NULL
int pthread_join(pthread_t thread, void **retval); // retval = NULL
int sleep(int seconds); // calling thread sleeps for a number of seconds
```

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

Idee: 1. Thread array mit threads füllen (über argv iterieren)
2. sort() thread methode mit sleep und print
3. Threads joinen

```
void *sort(void *arg) {
    int n = *((int*) arg);
    sleep(n);
    printf("%d", n);
    return NULL;
}
int main(int argc, char *argv[]) {
    pthread_t threads[argc]; int i;
    for (i=0; i<argc; i++) { if(i==0) continue;
        ~~pthread_create(threads+i, NULL, sort, atoi(argv[i]))~~
        int n = atoi(argv[i]);
        pthread_create(threads+i, NULL, sort, (void*) &n);
    }
    for (i=0; i<argc; i++) { if(i==0) continue;
        pthread_join(threads[i], NULL);
    }
    return 0;
}
```

IPC mit Pipes

2) Schreiben Sie ein Programm, das eine Pipe vom Enkel zum Grandparent Prozess erstellt.

Der Enkel soll die Nachricht "hello" zum Grandparent schicken, über die Pipe. Punkte: 10/ 12

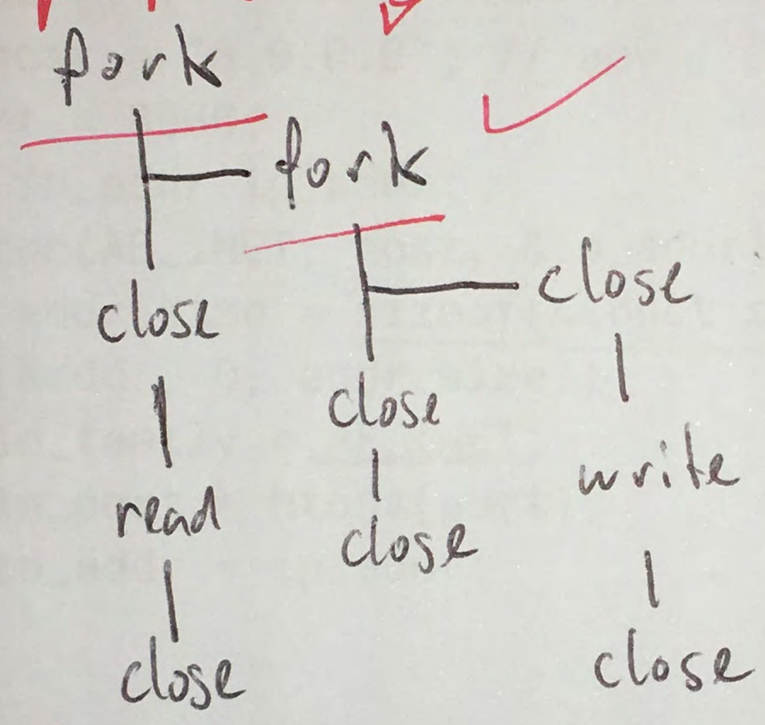
Hier ein Auszug aus der Doku, #includes und Fehlerbehandlung können Sie weglassen:

```
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.
pid_t wait(int *status); // wait for child process, status can be NULL
ssize_t write(int fd, const void *buf, size_t count); // write to a file
```

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

pipe

Idee:




```
int main() {
  int fd[2]; pipe(fd); char buf[5];
  if (fork()==0) {
    close(fd[0]); close(fd[1]); wait(NULL);
    if(fork()==0) {
      close(fd[0]);
      write(fd[1], "hello", 5);
      close(fd[1]);
    }
  } else {
    close(fd[1]);
    read(fd[0], buf, 5);
    close(fd[0]);
  } return 0;
}
```

Sockets

3) Gegeben den folgenden Code, implementieren Sie in *main()* einen HTTP Server, der Web Requests (Länge < MAX_REQ_LEN) liest und auf STDOUT_FILENO ausgibt. Punkte: 16 / 18

```
#define BACKLOG 5
#define MAX_REQ_LEN 4096

struct sockaddr_in addr;
char request_buf[MAX_REQ_LEN];
char response_buf[] =
    "HTTP/1.1 200 OK\r\n"
    "Connection: close\r\n"
    "Content-Length: 0\r\n"
    "\r\n";

void initAddr() {
    char *host = "0.0.0.0"; // any
    int port = 8080;
    struct in_addr ip_addr;
    inet_pton(AF_INET, host, &ip_addr);
    size_t addr_size = sizeof(struct sockaddr_in);
    memset(&addr, 0, addr_size);
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr = ip_addr;
}

// TODO: implement int main() { ... }
```

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

```
int accept(int sock_fd, struct sockaddr *addr, socklen_t *addrlen); //
  accept a connection on a socket, returns a (client) socket descriptor
int bind(int sock_fd, const struct sockaddr *addr, size_t addr_size); //
  bind a local address to a socket, returns 0 on success
int close(int fd); // close a file descriptor, returns 0 on success
int listen(int sock_fd, int backlog); // listen for connections on a
  socket, backlog = number of pending connections, returns 0 on success
ssize_t read(int fd, void *buf, size_t count); // read from a file descr.
int socket(int domain, int type, int pr); // domain = AF_UNIX or AF_INET,
  type = SOCK_STREAM or SOCK_DGRAM, pr = 0, returns a socket descriptor
ssize_t write(int fd, const void *buf, size_t count); // write to a file
```

(Fortsetzung auf der nächsten Seite)

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

Idee:
1. socket(AF_INET, SOCK_STREAM, ..)
2. bind()
3. listen()
4. while (accept())
5. read()
6. write()
7. close()

```
int main() {
    int sfd, afd;
    initAddr();
    sfd = socket(AF_INET, SOCK_STREAM, 0);
    bind(sfd, (sockaddr*) &addr, sizeof(addr));
    listen(sfd, BACKLOG);
    while ((afd = accept(sfd, (sockaddr*) &addr, sizeof(addr))) {
        read(afd, request_buf, MAX_REQ_LEN);
        write(STDOUT_FILENO, request_buf, MAX_REQ_LEN);
        close(afd);
    }
}
```

4) Welches Problem wird durch das Festlegen einer *Network Byte Order* gelöst? Punkte: 2 / 3

Die Art, in welcher Reihenfolge die übertragenen Bytes gelesen werden → Big Endian für Network Byte Order

Terminals

5) Welcher Terminal-Mode wird von der Shell verwendet, und wieso? Punkte: 3 / 3

Canonical mode, weil Input Zeilenweise verarbeitet wird, mit anderen Worten: Immer erst nachdem Enter gedrückt wird.

POSIX IPC

6) Nennen Sie zwei Unterschiede zwischen *named* und *unnamed* Semaphoren. Punkte: 4 / 4

Named POSIX Semaphore	Unnamed POSIX Semaphore
Kann Prozessübergreifend geshared werden	Kann nur mit verwandten Prozessen geshared werden
Lebt über Prozesslebenszeit hinaus	Wird spätestens nach Prozess Ende zerstört

Ein Sempahor soll die Anzahl Threads konstant halten. Punkte: 18

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

```
void free(void *ptr); // free memory allocated on the heap
void *malloc(size_t size); // allocate size bytes of memory on the heap
int printf(const char *format, ...); // format string %s, char %c, int %d
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
  void *(*start) (void *), void *arg); // starts a thread; attr = NULL
int pthread_detach(pthread_t thread); // detach a thread, no join needed
int rand(); // returns a random number [0, RAND_MAX]
int sleep(int seconds); // calling thread sleeps for a number of seconds
```

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

Idee: while (1)
sem_wait
~~start thread~~
~~detach thread~~ to avoid memory leak

thread:
sleep random
sem_post

Global:
sem_t

Init:
sem_init(..., n)

Annahme n wird als CLI Argument geliefert.

```
sem_t sem;
int main(int argc, char *argv[]) {
    int n;
    pthread_t t;
    n = atoi(argv[1]);
    sem_init(&sem, 0, n);
    while (1) {
        sem_wait(&sem);
        pthread_create(&t, NULL, handle, NULL);
        pthread_detach(t);
    }
}
```

Funktion muss vor main

```
void *handle(void *arg) {
    sleep(rand());
    sem_post(&sem);
    return NULL;
}
```

~~t = (pthread_t *)arg~~
~~free(t)~~

Zeitmessung

8) Schreiben Sie ein Programm das misst, wie lange es dauert, eine 1-Byte Message durch eine POSIX Message Queue mit Namen "/mq" an sich selbst zu schicken.

Punkte: 16 / 16

```
$ ./my_mq_msg_time
real = 47000 nsec
```

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

```
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; long tv_nsec; }; // time in s, nano
mqd_t mq_open(char *name, int flag, mode_t mode, struct mq_attr *attr);
  // open a message queue, flag = O_RDWR|O_CREAT, mode = S_IRUSR|S_IWUSR,
  // struct mq_attr { long mq_maxmsg; long mq_msgsize } // mq attributes
ssize_t mq_receive(mqd_t mq, char *msg, size_t msg_len, int *msg_prio);
int mq_send(mqd_t mq, char *msg, size_t msg_len, int msg_prio);
int mq_unlink(char *name); // remove a message queue
int printf(const char *frmt, …); // frmt int %d, long %ld, string % ...
```

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

Idee: mq_open
measure time
send
receive
measure time
print delta
mq_unlink

Global:
struct timespec t0, t1;

(✓) Default attr according to man pages

```
int main() {
  mqd_t mq; struct timespec t0, t1; char b;
  mq = mq_open("/mq", O_RDWR|O_CREAT, S_IRUSR|S_IWUSR, NULL)
  clock_gettime(CLOCK_REALTIME, &t0);
mq_send → mq_send(mq, "0", 1, 1);
  mq_receive(mq, &b, 1, NULL);
  clock_gettime(CLOCK_REALTIME, &t1);
  printf("real = %ld nsec \n", (t1.tv_sec-t0.tv_sec)*1000000000
                                + t1.tv_nsec-t0.tv_nsec);
  mq_unlink("/mq");
  return 0;
}
```