SP1 Miniklausuren

hu78sapy@stud.cs.fau.de

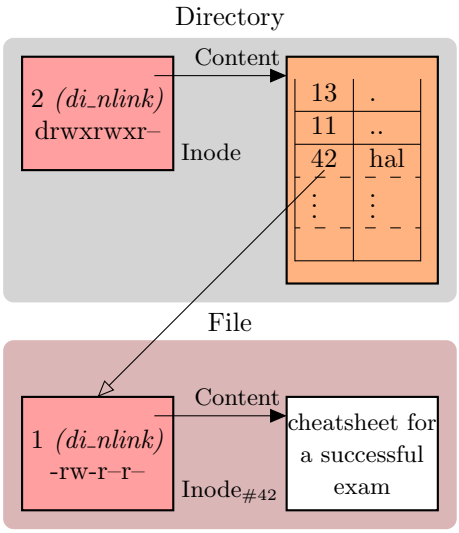
23. April 2013

1 14. Juni 2010

1.1 Aufgabe 1

a) b
b) c
c) a

1.2 Aufgabe 3



1.3 Aufgabe 2

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

/*--Extra--*/
#include <stdio.h>

/*--Necessary for waitpid--*/
#include <sys/types.h>
#include <sys/wait.h>

static int mycomp(const void* fir, const void* sec) {
        return strcmp(*(char**) fir, *(char**) sec);
}

int sorted_exec(char* prg, char* args[]) {
        /*--Get args length--*/
        size_t counter=0;
        while(NULL!=args[counter]){counter++;}
#if 0
        size_t counter=0;
        for(char* ptr=args[counter];NULL!=ptr;ptr=args[++counter]){}
#endif

        /*--Sort array--*/
        qsort(args,counter,sizeof(char*),mycomp);

        /*--Execute each argument in one thread--*/
        int ret=0;
        pid_t pid;
        for(int x=0;x<counter;x++) {
                pid = fork();
                if(0>pid) {return -1;}
                if(0==pid) {
                        execlp(prg,prg,args[x],NULL);
                        exit(EXIT_FAILURE);
                }
                int status;
                if(-1==waitpid(pid,&status,0)) {return -1;}
                if(WIFEXITED(status) && WEXITSTATUS(status) != 0) {ret=-1;}
        }
        return ret;
}

int main(int argc, char* argv[]) {
        char* prgname = "/bin/ls";
        char* args[3];
        args[0]="..";
        args[1]=".";
        args[2]=NULL;
        int status=sorted_exec(prgname,args);
        printf("status: %d\n",status);
}
```

2 08. Dezember 2010

2.1 Aufgabe 1

a) b
b) a
c) c

2.2 Aufgabe 3

Eine Semaphore ist ein abstrakter Datentyp zur Signalisierung von Ereignissen zwischen gleichzeigiten Prozessen. Sie hat zwei unteilbare Operationen:

P hat der Semaphor den Wert 0, wird der laufende Prozess blockiert, ansonsten wird der Semaphor um 1 dekrementiert.

V inkrementiert den Semaphor um 1, auf den Semaphor ggf. blockierte Prozesse werden deblockiert.

2.3 Aufgabe 2

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>

//TODO: line is const, how to split?
int execute(const char* line) {
        /*--Split line into cmd+args--*/
        char copy[strlen(line)+1];
        strcpy(copy,line);
        char* argv[strlen(copy)+1];
        char* ptr=strtok(copy,"\t ");

        int pos=0;
        while(NULL!=ptr) {
                argv[pos] = ptr;
                pos++;
                ptr=strtok(NULL,"\t ");
        }
        argv[pos]=NULL;

        /*--Execute Commando in new process--*/
        pid_t pid = fork();
        if(0>pid) {return -1;}
        if(0==pid) {
                execvp(argv[0],argv);
                exit(EXIT_FAILURE);
        }

        /*--Father--*/
        int status;
        if(-1==waitpid(-1,&status,0)) {return -1;}
        if(!WIFEXITED(status)) {return -2;}
        return WEXITSTATUS(status);
}
int main(int argc, char* argv[]) {
        char line[40];
        strcpy(line,"/bin/ls .");
        int status = execute(line);
        printf("Exit: [%d]\n",status);
        exit(EXIT_SUCCESS);
}
```

3 10. Mai 2011

3.1 Aufgabe 1.1

a) b
b) c

3.2 Aufgabe 1.2

a) a,c,d

3.3 Aufgabe 3

schwergewichtige Prozesse Horizontale Isolation von anderen Fäden/allgemeinen Programmadressräumen

leichtgewichtige Prozesse Vertikale Isolation vom Betriebssystemadressraum

federgewichtige Prozesse Keine Isolation, der reine Kontrollfluss: Faden

3.4 Aufgabe 2

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <errno.h>
#include <string.h>

int newer(DIR *dir, char* refFile, char* files[], int maxFiles) {
        /*--Get reference file modification date--*/
        printf("Get ref..\n");
        int nFiles=0;
        struct stat buf;
        if(-1==stat(refFile,&buf)) {return -1*nFiles;}
        int rmod=buf.st_mtime;

        /*--Traverse directory--*/
        struct dirent* cur;
        char file[4096];
        errno=0;
        while((NULL!=(cur=readdir(dir)))&&(maxFiles>=nFiles)) {
                //TODO: how to get dirname of DIR?
                sprintf(file,"%s/%s","mydir",cur->d_name);

                if(-1==stat(file,&buf)) {return -1*nFiles;}
                int fmod=buf.st_mtime;

                /*--Check if modification date of file is newer than mod date of ref-file--*/
                if(fmod<rmod) {
                        strcpy(files[nFiles],file);
                        nFiles++;
                }
        }

        /*--Extra: Print all new files--*/
        for(int x=0;x<nFiles;x++) {printf("[%d]: %s\n",x,files[x]);}

        /*--Return #files--*/
        if(0!=errno) {return -1*nFiles;}
        return nFiles;
}

int main(int argc, char* argv[]) {
        int maxFiles=10;
        char** files = (char**) malloc(10*sizeof(char*));
        if(NULL==files) {exit(EXIT_FAILURE);}
        for(int i=0; i<maxFiles; i++) {
                files[i] = (char*) malloc(sizeof(char));
                if(NULL==files[i]) {exit(EXIT_FAILURE);}
        }
        char* refFile="./refFile";
        DIR* dir = opendir("./mydir");

        int status=newer(dir,refFile,files,maxFiles);
```

```
        printf("status: [%d]\n",status);
        exit(EXIT_SUCCESS);
}
```

4 26. Oktober 2011

4.1 Aufgabe 1.1

a) c
b) d

4.2 Aufgabe 1.2

a) c,d,?

4.3 Aufgabe 3

4.3.1 3.a

	synchron	vorhersagbar	reproduzierbar	Quelle
Trap	✓	✓	✓	unbekannter Befehl, falsche Adressierungsart oder Rechenoperation
Interrupt	×	×	×	Signalisierung externer Ereignisse, Beendigung einer DMA- bzw E/A-Operation, Seitenfehler im Falle globaler Ersetzungsstrategien

4.3.2 3.b

Wiederaufnahmemodell *en.: resumption model*

- Behandlung der Ausnahmesituation führt zur **Fortsetzung** der Ausführung des unterbrochenen Programms
- Trap kann, Interrupt muss

Beendigungsmodell *en.: termination model*

- Behandlung der Ausnahmesituation führt zur **Abbruch** der Ausführung des unterbrochenen Programms
- Trap kann, Interrupt darf niemals

4.4 Aufgabe 2

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include "sem.h"

void P(SEM* s);
void V(SEM* s);

static int sum;
static SEM* sem;

int psum(const char* dirName) {
        /*--Create and initialize semaphore--*/
        sem=semCreate(1);
        if(NULL==sem) {return -1;}

        /*--Traverse directory--*/
        DIR* dir = opendir(dirName);
        if(NULL==dir) {return -1;}

        struct dirent* cur;
        char file[4096]; //TODO, whats max?
        struct stat buf;
        while(NULL!=(cur=readdir(dir))) {
                if(-1==lstat(cur->d_name,&buf)) {return -1;}
                if(!S_ISREG(buf.st_mode)) {continue;}
//                sprintf(file,"%s/%s",dirName,cur->d_name); // nicht erlaubt
                strcpy(file,dirName);
                strcat(file,"/");
                strcat(file,cur->d_name);

                P(sem);
                sum += buf.st_size;
                V(sem);
        }
        if(-1==closedir(dir)) {return -1;}
        if(-1==semDestroy(sem)) {return -1;}
        return 0;
}
int main(int argc, char* argv[]) {
        psum(".");
        exit(EXIT_SUCCESS);
}
```

5 19. April 2012

5.1 Aufgabe 1.1

a) b
b) d

5.2 Aufgabe 1.2

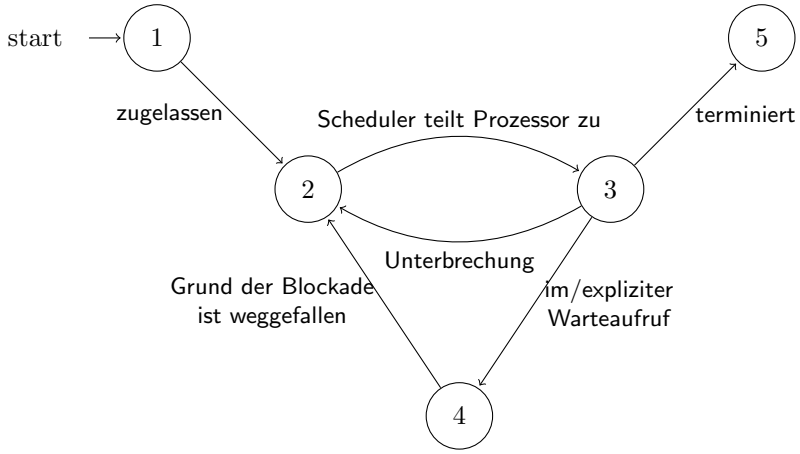
a) d,e,f

5.3 Aufgabe 3

Prozesszustände:

1. Erzeugt (*en. New*) : Prozess wurde erzeugt. Betriebsmittel stehen für Prozess noch nicht bereit.
2. Bereit (*en. Ready*) : Prozess ist bereit zum laufen. Betriebsmittel sind vorhanden.
3. Laufend (*en. Running*) : Prozess wird vom Prozessor ausgeführt.
4. Blockiert (*en. Blocked/Waiting*) : Prozess wartet auf ein Ereignis und wird zum Warten blockiert.
5. Beendet (*en. Terminated*) : Prozess ist beendet. Jedoch muss er aus bestimmten Gründen im System verbleiben (Z.b. Betriebsmittel sind noch nicht freigegeben).

Zeichnung:



5.4 Aufgabe 2

```c
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <stdio.h>
#include <errno.h>

struct jobDesc {
        pthread_t tid;
        char input[101];
        int output;
        int error;
};

struct jobDesc* multithread(char* filename, int numLines, void (*threadFunc)(struct jobDesc*)) {
        struct jobDesc* job = (struct jobDesc*) malloc(numLines*sizeof(struct jobDesc));
        int max=100;
        char buf[max+1];
        FILE* file = fopen(filename, "r"); //other rights?
        if(NULL==file) {return NULL;}
        for(int i=0; errno=0, NULL!=fgets(buf,(max+1),file); i++) {
                strcpy(job[i].input,buf);
                errno = pthread_create(&(job[i].tid), NULL, (void* (*) (void*))threadFunc, &(job[i]));
                if(0!=errno) {job[i].error=errno;}
        }
        if((ferror(file)&&fclose(file)) != 0) {return NULL;}
        return job;
}

static void mywc(struct jobDesc* job) {
        job->output = strlen(job->input);
        return;
}

int main(int argc, char* argv[]) {
        char* filename = "./myfile";
        int numLines = 10;
        struct jobDesc* jobs = multithread(filename,numLines,mywc);
        for(int i=0; i<numLines; i++) {
                errno=pthread_join(jobs[i].tid,NULL);
                if(errno) {perror("pthread_join"); exit(EXIT_FAILURE);}
        }
        for(int i=0; i<numLines; i++) {
                printf("[%lu] Input: %s",(unsigned long int)jobs[i].tid,jobs[i].input);
                printf("[%lu] Output: %d, Error: %d\n",(unsigned long int)jobs[i].tid,jobs[i].output,jobs[
        }
        free(jobs);
        exit(EXIT_SUCCESS);
}
```

6 25. Oktober 2012

6.1 Aufgabe 1.1

a) d
b) b

6.2 Aufgabe 1.2

a) a,d,f

6.3 Aufgabe 3

6.3.1 3.a

Eine Semaphore ist ein abstrakter Datentyp zur Signalisierung von Ereignissen zwischen gleichzeigiten Prozessen. Sie hat zwei unteilbare Operationen:

P hat der Semaphor den Wert 0, wird der laufende Prozess blockiert, ansonsten wird der Semaphor um 1 dekrementiert.

V inkrementiert den Semaphor um 1, auf den Semaphor ggf. blockierte Prozesse werden deblockiert.

6.3.2 3.b

Hauptthread:	**Arbeiter-Thread:**
sem = sem_init(0);	doWork();
startWorkerThread();	V(sem);
P(sem);	
P(sem);	
P(sem);	
P(sem);	
P(sem);	

6.4 Aufgabe 2

```
#include <dirent.h>
#include <errno.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>

struct Entry {
        char name[NAME_MAX+1];
        off_t size;
};

static int mycomp(const void* fir, const void* sec) {
#if 0
        struct Entry* fE = *((struct Entry**) fir);
        struct Entry* sE = *((struct Entry**) sec);
#endif
        off_t fsize = ((struct Entry*) fir)->size;
        off_t ssize = ((struct Entry*) sec)->size;
        if(fsize<ssize) {return -1;}
        if(fsize>ssize) {return 1;}
        return strcmp(((struct Entry*) fir)->name,((struct Entry*) sec)->name);
}

int getSortedEntries(struct Entry entries[], unsigned int maxEntries) {
        /*--Traverse directory--*/
        DIR* dir = opendir(".");
        if(NULL==dir) {return 0;}

        errno=0;
        struct dirent* cur;
        int nFiles=0;

        while((nFiles<maxEntries) && (NULL!=(cur=readdir(dir)))) {
                char file[PATH_MAX];
                sprintf(file,"%s/%s",".",cur->d_name);
                strcpy(entries[nFiles].name,file);
                struct stat buf;
                if(-1==lstat(file,&buf)) {return 0;}
                entries[nFiles].size=buf.st_size;
                nFiles++;
        }

        /*--Check for errors, otherwise return nFiles--*/
        if(0!=errno) {return 0;}
        if(NULL!=(cur=readdir(dir))) {return -1*nFiles;}
        qsort(entries,nFiles,sizeof(struct Entry),mycomp);
        if(0!=closedir(dir)) {return 0;}
        return nFiles;
}

int main(int argc, char *argv[]){
        int maxFiles=10;
        struct Entry entries[maxFiles];
```

```
        int ex = getSortedEntries(entries, maxFiles);
        printf("return : %d\n", ex);
        return 0;
}
```

7 Cheatsheet

7.1 Codeschnipsel

```
/*--compare function--*/
static int mycomp(const void* fir, const void* sec) {
        return strcmp(*(char**) fir, *(char**) sec); //je nach situation...
}

/*--qsort--*/
/** args: array, which is to be sorted
        * nmemb: #elements of array
        *       size: sizeof(element)
        *       compar: compare function
        **/
qsort(void* args, size_t nmemb, size_t size,int (*compar)(const void*, const void*));

/*--fork--*/
/**
        * pid: contains pid of child for father
        **/
pid_t pid = fork();
if(0>pid) {/*ERROR*/}
if(0==pid) {/*CHILD*/}
/*FATHER*/

/*--exec [Returns only if error]--*/
execl(prgpath,prgpath,args1,...,NULL); //path
execlp(prgname,prgname,args1,...,NULL); //file
execv(prgpath,args); //args is NULL-terminated array, starts with prgpath
execvp(prgname,args); //args is NULL-terminated array, starts with prgname

/*--waitpid [Waiting for a child]--*/
/** List of status evaulation macros:
        * WIFEXITED(status) : @return 1 if normal exit
        * WEXITSTATUS(status) : @return exit status of child (use only if WIFEXITED was true...)
        * WIFSIGNALED(status) : @return 1 if child process terminated by a signal
        * ...
        **/
int status;
if(-1==waitpid(pid,&status,0)) {/*ERROR*/}

/*--Get file status--*/
/** List of contents: [*]
        *       st_mode rights [*]-check with [**]
        *       st_size size of file
        *       st_(a,m,c)time time of last (acces,mod,status change)
        *       Access with buf.st_*
        **/
/** [**]
        *
        *       S_ISREG(buf.st_mode) : reg file
        *       S_ISDIR(buf.st_mode) : directory
        */
struct stat buf;
if(-1==lstat(file,&buf)) {/*ERROR*/}
```

```
/*--Directory traversal--*/
/** contents of cur
        *       d_name : filename
        **/
char* path="/path/to/file";
DIR* dir = opendir(path);
if(NULL==dir) {/*ERROR*/}
struct dirent* cur;
while(errno=0,NULL!=(cur=readdir(dir))) {/*CODE*/} //maybe check with [*]-[**]
if(0!=errno) {/*ERROR*/}
if(0!=closedir(dir)) {/*ERROR*/}

/*--Semaphores--*/
static SEM* sem;
// {...
        sem = semCreate(/*#ACCESS*/);
        if(NULL==sem) {/*ERROR*/}
        P(sem);
        /*syn. CODE*/
        V(sem);
        if(-1==semDestroy(sem)) {/*ERROR*/}
// ...}

/*--Open file stream--*/
FILE* file = fopen(filename,"a+");
if(NULL==file) {/*ERROR*/}
char buf[size];
while(NULL!=fgets(buf,size,file)) {/*CODE*/} // size is something you have to determine
if(ferror(file)) {/*ERROR stream*/}
if(fclose(file)) {/*ERROR*/}

/*--Thread Creation/Detaching/Joining--*/
pthread_t tid; // store tid info for join/detach, if not used: tid can be NULL in pthread_create
errno = pthread_create(tid,NULL,(void* (*) (void*))threadFunc, args); //args is argument of threadFunc
if(0!=errno) {/*ERRNO*/}
//...
for(/*each thread*/) {
        errno=pthread_join(tid,NULL);
        if(errno) {/*ERROR*/}
}

/**--string.h--**/
/*--strtok [tokenize a string]--*/
char* ptr=strtok(args,delim);
while(NULL!=ptr) {/*CODE*/ptr=strtok(NULL,delim);}
/*--strcpy [copy a string]--*/
char copy[strlen(orig)+1];
strcpy(copy,orig);

/*--Other--*/
sprintf(char* str, const char* format,...);
snprintf(char* str, size_t size, const char* format,..);
//example:
sprintf(file,"%s/%s",path,cur->d_name);
```