Operating System
Assignment 6
Design Document

The files we changed are :

- Thread.h
- Thread.c
- Signal.h
- Signal.c

Firstly the following data structures were changed in the pre existing thread.c and thread.h files. The thread structure was changed and following fields were added :

```
    tid_t ppid;                 //parent threads pid
    int chld;                   //total number of children of a thread
    int rem;                    //number of children alive
    int lifetime;               //for counting the waiting + running ticks of a thread
    unsigned short mask;        //for signal mask of each thread defining which signals to
//                              //ignore and which signals should call their default signal
                                //handlers
```

Some new functions were also introduced to help out in signal.c :

```
struct thread * thread_foreach2(tid_t );  //iterates over all threads alive and finds the
//                                        //thread object given its unique pid.

void thread_exit2 (struct thread *t) NO_RETURN;      //to exit a particular thread object
```

A macro to determine the max lifetime of threads was also defined in thread.h file :

```
#define MAXLIFE 500
```

Now, coming to thread.c file :

The following functions were added apart from thread_exit2 and thread_foreach2 as explained in the above section :

```
int alive(tid_t p)
     {
       struct list_elem *e;
       for (e = list_begin (&all_list); e != list_end (&all_list);
            e = list_next (e))
         {
           struct thread *t = list_entry (e, struct thread, allelem);
           if(t->tid == p)
             return 1;
         }
       return 0;
     }
```

****

```
void setlifetime(struct thread* t)    {
          t->lifetime=MAXLIFE;
     }
```

****

++

The following functions were modified in thread.c :

```
thread_init()  :                    //Initialising the signal list defined in signal.h
     list_init (&signal_list);
     printf("Signal list initialised\n");
```

****

```
thread_ticks() :                    //To increment the life ticks of all threads

       struct list_elem *e;

         for (e = list_begin (&all_list); e != list_end (&all_list);
              e = list_next (e))
           {
                struct thread *t1 = list_entry (e, struct thread, allelem);
                if( t1->lifetime > 0 && t1 != idle_thread ) t1->lifetime--;
               }
```

****

```
thread_exit2() :                    //To add the SIG_CHLD signals
            if(alive(thread_current()->ppid))
              {
                signal *s=(signal *)malloc(sizeof(signal));
                s->sig=SIG_CHLD;
                s->pid=thread_current()->ppid;
                s->cid=thread_current()->tid;
                list_push_back(&signal_list,&(s->signal_elem));
              }
```

****

```
thread_schedule_tail() :      //To call the signal handlers, it ensures that the signal
/                             //handlers(default) are called after each context switch after
interrupt enables
           struct thread *t=thread_current();
             if (t->lifetime ==0 ) {
               if( (t->mask%2 )==0) SIG_CPU_handler(cur);
             }
             handler(t);
```

****

```
Init_thread () :                    //initialising new elements of the thread structure
             t->mask=0;
             setlifetime(t);
```

****

The functions and data structures for signal.h and signal.c are defined as below along with their design flows

The contents of signal.h are as follows :
A structure for signal, sigset and a signal list

```
void handler(struct thread *);                  //default all-signal handler which later calls other
handlers
void SIG_CPU_handler(struct thread *);  //SIG_CPU default handler
void SIG_KILL_handler(struct thread *,signal *); //SIG_KILL default handler
void SIG_USR_handler(signal *);            //        default handler for SIG_USR
void SIG_CHLD_handler(struct thread *,signal *);// default handler for SIG_CHLD
void SIG_UNBLOCK_handler(signal * );             //        default handler for SIG_NBLOCK

int Signal(enum sigtype ,enum action );    //sets the signal mask of current thread
int kill(enum sigtype ,tid_t );          //        function to pass signal to other thread

int sigemptyset(sigset_t *);            //        initialising mask to empty
int sigfillset(sigset_t *);                //        initialising mask to full
int sigaddset(sigset_t *,enum sigtype);                        //adding signal to mask
int sigdelset(sigset_t *,enum sigtype);                         //deleting signal from the mask
int sigprocmask(enum how,sigset_t *, sigset_t *);         //        function for setting mask
```

For handling the signals we are calling the handler function in the end of schedule_tail() function.

The handler function traverses the global signal list and finds all the signals to be delivered to the current thread. It then calls the corresponding signal handlers for that signals. Here before calling the signals the mask value is checked that may be set by Signal() function or sigprocmask() function to find out whether the signal is blocked for the current process or not.

The corresponding signal handlers are:

SIG_CHLD_handler : it will print the id of thread exited, no. of threads created by the parent thread and number of threads alive at that time

SIG_USR_handler : it will print the id of the thread to which the signal is sent and the id of the thread who sent the signal

SIG_KILL_handler: it will print which thread sent the kill signal and will then call thread_exit2() to kill the thread.

SIG_CPU_handler: it will print that the lifetime of thread expired and then kill the thread by calling thread_exit2()

SIG_UNBLOCK_handler : this handler will be called for each SIG_UNBLOCK signal irrespective of which thread it is meant for. The handler will first assert that the thread is blocked and if it is blocked it will call thread_unblock() to unblock it.

Other functions added are:

Signal(enum sigtype a,enum action b) : this function will set the action b for signal type a for the thread calling. B can be SIG_IGN OR SIG_DFL.

kill(enum sigtype a,tid_t b) : kill() is used to send a signal to other thread. The a signal will be sent to thread with id b by the running thread.

For signal masking we created a sigset_t structure that contained a unsigned short int whose bits we used to check masking.

Whenever a sigaddset() or sigdelset() is called the mask of the process is changed according to the new sigset_t structure passed. sigaddset() required or-ing with a particular bit stream while sigdelset() required first and of new set with existing mask and then xor of result with existing mask.

For sigemptyset() the short int is initialised to 0 while for sigfillset() the short int is initialised to 255;

For the sigprocmask() the existing mask is modified to the set passed according to the mode passed which can be SIG_BLOCK,SIG_UNBLOCK and SIG_SET.

Written by :

Apurv Kumar (14CS10006)
Shubham Sharma (14CS30034)