ΕΡΓΑΣΙΑ 2

global μεταβλητες mythreads.h

typedef struct thread_s{
  int out_of_order; int out_reason;
  ucontext_t cont; int thr_name;
  struct thread_s *next;
}thr_t;
	typedef struct sems{
	  int semid; int val;
	  int locked; int last_up;
	}sem_t;
		thr_t *header;
		ucontext_t *curr_co, *next_co;
		int curr_name;
		sigset_t set;

main

```
For{ sem_init(workerSems[i], 0);   }
sem_init(mtx, 1);
sem_init(waitingMain, 0);
init(&thr_main);     curr_name = 0;
create(&thr_handler,  handler,  &thr_handler);
next_co = &thr_handler.cont;
sigprocmask(UNBLOCK);
for{  scanf( );  flag++;
     sigprocmask(BLOCK);
     create(&worker_thr[i], primesearch, workers[i]);
     sigprocmask(UNBLOCK);
}
while(1){
      scanf();  sem_down(waitingMain);
      for(find worker[i]){  sem_up(workerSem[i]);}
}
terminatingSequence = 1;
for{
      num = -1;  sem_down(waitingMain);
      for(find worker[i]){  sem_up(workerSem[i]);}
}
sigprocmask(BLOCK);
for( ){
   join(&worker_thr[i]);
}
for( ){
  destroy(&worker_thr[i]);
}
termination = 1;
join(&thr_handler);
κανουμε τα sem_destroy(...);
```

primesearch

```
if(isfirst!=1){
  sem_up(mtx);
  sem_down(workerSems[temp->onoma]);
  if(temp->number==-1){
    sem_down(mtx);    terminated[temp->onoma]=1;
    waitingWorkers--;
    if(waitingWorkers>0
          &&terminatingSequence==1){
      sem_up(waitingMain);
    }
    sem_up(mtx);   return;
  }
  sem_down(mtx);
  waitingWorkers--;
  if(waitingWorkers>0){
     sem_up(waitingMain);
  }
  sem_up(mtx);
}
else{   isfirst=0;  }
sem_down(mtx);
primetest(temp->number );
waitingWorkers++;   temp->isready = 1;
if(waitingWorkers==1){
    sem_up(waitingMain);
}
```

handler

```
Search = curr_thr;
do{
  search = search->next;
  while(search->out_of_order!=0){
        search = search->next;
  }
}while(search == handler);
mycoroutines_switchto(next_co,&search->cont);
for{
  if(terminated[k]==1){
      search->out_of_order = -1;
  }
```

Mythreads.c

static void SIGALRM_handler(int sig)

```
sigprocmask(BLOCK);
search = curr_thr;
mycoroutines_switchto(&search->cont,next_co);
sigprocmask(UNBLOCK);
```

int mythreads_init(thr_t *thr_main)

```
mycoroutines_init(&thr_main->cont);
header=thr_main;
thr_main->next=header;
thr_main->out_of_order = 0;
```

```
  search=search->next;
}
if(termination == 1){
   break;
}
```

```
thr_main->thr_name = thr_name_counter;
thr_name_counter++;
Αρχικοποιηση SIGALARM και αρχκοιπηση σετ
για blocking και unblocking του σηματος
```

Mythreads.c

int mythreads_create(thr_t *thr,void func(), void *arg)

```
if(thr_name_counter==1){
    mycoroutines_create(&thr->cont, func,
                                arg,curr_co);
}
else{
    search=header->next;
    mycoroutines_create(&thr->cont,func,
                                arg,&search->cont);
}
thr->next = header;       thr->out_of_order = 0;
thr->out_reason = 0;      thr->thr_name =
thr_name_counter;         thr_name_counter++;
search = header;
while(search->next != header){
            search = search->next;
}
search->next = thr;
```

Mythreads.c

int mythreads_yield( )

```
sigprocmask(BLOCK);
Επαναρχικοποιηση του  SIGALARM
search = curr_thr;
mycoroutines_switchto(&search->cont,next_co);
sigprocmask(UNBLOCK);
```

int mythreads_join(thr_t *thr)

```
while(thr->out_of_order!=-1){
    mythreads_yield();   sigprocmask(BLOCK);
}
```

int mythreads_destroy(thr_t *thr)

```
search = header;
while(search->next->thr_name != thr->thr_name){
            search = search->next;
}
temp = search->next;     search->next = temp->next;
mycoroutines_destroy(&thr->cont);
```

Mythreads.c

int mythreads_sem_init(sem_t *sema, int value)

```
sema->locked=0;     sema->val = value;
sema->semid = semid_counter;
sema->last_up = 0;      semid_counter++;
```

int mythreads_sem_down(sem_t *sema)

```
while(sema->locked !=0){}
sema->locked=1;
if(sema->val > 0){
   sema->val--;   sema->locked=0;   return(0);
}
search = curr_thr;
search->out_of_order = 1;
search->out_reason = sema->semid;
sema->locked = 0;
mythreads_yield();
```

int mythreads_sem_destroy(sem_t *sema)

```
if(sema->val > 0){  free(sema);  return(0);  }
Διαφορετικα ελεγχει αν χρησιμοποιειται ακομα και
εμφανιζει μηνυμα λαθους
```

Mythreads.c

int mythreads_sem_up(sem_t *sema)

```
while(sema->locked !=0){}
sema->locked=1;
if(sema->val > 0){
   sema->val++;   sema->locked=0;   return(0);
}
search = το next απο αυτο που ξυπνησε τελευταιο
while(search->out_reason != sema->semid){
  if(search->next->thr_name== sema->last_up){
     if(search->next->out_reason != sema->semid){
           sema->val++;
           sema->last_up = header->thr_name;
           sema->locked=0;    return(0);
     }
     else{  search=search->next;   break; }
  }
  else{ search=search->next;  }
}
search->out_reason = 0;  search->out_of_order = 0;
sema->last_up = search->thr_name;
sema->locked=0;
```

Global task2.c	**Global mythreads.c**
int waitingWorkers, times, flag, termination;	struct sigaction act={{0}};
sem_t **workerSems, *mtx, *waitingMain;	struct itimerval t={{0}};
int *terminated, terminatingSequence;	int semid_counter = 1000;
	int thr_name_counter = 0;