Τέταρτη Εργασία Ταυτόχρονου Προγραμματισμού

Ομάδα: 11η
Μέλη:
Κωφόπουλος-Λυμπέρης Σταμάτης 2548
Παυλίδης Παναγιώτης 2608

Άσκηση 4: coroutines.c/.h

```
mycoroutines_init(){
   getcontext(main-context);}
mycoroutines_create(){
   getcontext(new_co);
   new_co->us_stack.ss_sp = malloc(SIGSTKSZ);
   new_co->uc_stack.ss_size = SIGSTKSZ;
   new_co->uc_link = link_context;
   makecontext(new_co, body, 1, arg);}
mycoroutines_switch(){
   swapcontext(old_co, new_co);}
mycoroutines_destroy(){
   free(co->uc_stack.ss_sp);
   co->uc_stack.ss_sp = NULL;}
```

```
void mycoroutines_init(ucontext *main_context, line);

void mycoroutines_create(ucontext *new_co, ucontext *link_context, void(*body)(), void *arg, line);

void mycoroutines_switch(ucontext *old_co, ucontext *new_co,  line);

void mycoroutines(ucontext *co, line);
```

Άσκηση 4.1 : Παραγωγός-Καταναλωτής

```
MAIN:
buffer_init();
input_fd = open(argv[1],RDONLY);
output_fd = open(argv[2], RDWR);

mycoroutines_init(main_context);
mycoroutines_create(&producer,&consumer,file_read,&input_fd);
mycoroutines_create(&consumer,&producer,file_write,&output_fd);
mycoroutines_switch(&main_context, &producer);

close(input_fd);   close(output_fd);
mycoroutines_destroy(&producer); (&consumer);
sprintf(diff, “diff %s %s”,argv[1], argv[2]);
system(diff);
mycoroutines_destroy(&main_context);
```

```
void file_read(){
while(1){
  check_read = read(*input_fd, &c, 1);
  if(check_read==-1){ERROR}   else if(check_read==0){break;}
  if(size<SIZE){buffer[size++] = c}
  else{mycoroutines_switch(&producer,&consumer); buffer[size++]=c;}
}
if(size>0){mycoroutines_switch(&producer, &consumer);
  mycoroutines_switch(&producer, &main_context);}
else{mycoroutines_switch(&producer, &main_context);}
}
```

```
void buffer_init{
for(i=0; i<SIZE; i++){
  buffer[i] = ‘\0’;}
}
```

```
void file_write(){
while(1){
  if(size>0){
    c = buffer[i++];  check_write = write(*output_fd,&c,1);
    size--;
  }
  else{ mycoroutines_switch(&consumer,&producer); i=0; }}}
```

Άσκηση 4.2: round-robin.c/.h

```
#include “coroutines.h”
typedef struct{int val; int id;}sem_t;
typedef struct list{ int state; int thread_id, int sem_id; ucontext_t context_id;
   struct list *next; struct *prev;}list_t;
extern list_t *head;   extern struct sigaction act;   extern struct itimerval timer;
extern ucontext_t handler_context;    extern sigset_t signal_mask;
void list_init();    list_t *list_search;
list_t *list_insert_first(ucontext_t context_id);    voidlist_delete_last();
int change_state(int thread_id, int val);    void scheduler();
```

```
void init_list(){   head = (list_*)malloc(sizeof(list_t));
          head->next  = head;    head->prev = head;}
list_t *list_search(){
     for(current = head->next; current != head; currnet = current->next)
     {thread = current;}    return(thread);}
list_t *list_insert_first(ucontext_t context_id){
  new= (list_t*)malloc(sizeof(list_t));    new->state=1;
  new->context_id=context_id;    new->next=head->next;
  new->prev = head;  new->next->prev = new;
  new_node->next->prev = new;    return(new); }
void list_delete_last(){
   thread = head->next;  thread->next->prev = thread->prev;
   thread->prev->next = thread->next;  free(thread);}
int change_state(int thread_id, int val){
  for(current=head->next; ((current!=head)&&
     (current->thread_id!=thread_id));current=current->next){}
  if(current!=head){current->state=val; return(0);}  else{return(-1);}}
```

```
void scheduler() {
while(1) {
  sigprocmask(SIG_BLOCK,&signal_mask,NULL);
  context = list_search();    state1 = context->state;
  do{   sigwait(&sigmask, &sig);
         state = context->state;   thread_context = context->context_id;
         sem_id = context->sem_id;    thread_id = context->thread_id;
     list_delete_last();
     if(state1==666) {mycoroutines_destroy(&thread_context,__LINE__);}
     else {  context = list_insert_first(thread_context);
               context->thread_id = thread_id;
               context->sem_id = sem_id; context->state = state;}
     for(current = head->next; current != head;current = current->next) {}
     stored_context = list_search();
  }while(context->state == 0);
  ready_context = context;  timer.it_value.tv_usec = 60;
  mycoroutines_switch(&handler_context,
    &(ready_context->context_id),__LINE__);}}
```

Άσκηση 4.2: thread.c/.h

```
int mythreads_join(ucontext_t thread,long line) {
list_t *current_context, *selected_thread; for(current_context = head->next;
  current_context != head; current_context = current_context->next) {
    if(current_context->state == 1) {selected_thread = current_context;}  }
selected_thread->join = 0;  change_state(selected_thread->thread_id,0);
for(current_context = head->next; current_context != head;
  current_context = current_context->next) {
if(&(current_context->context_id) == &thread) { current_context->join = 0; }}
mycoroutines_switch(&(selected_thread->context_id),&handler_context,line);
return(0); }
```

```
#include "round_robin.h"

void mythreads_init(long line);
int mythreads_create(ucontext_t thread,void(*body)(),void *arg,long line);
int mythreads_yield(long line);    int mythreads_join(ucontext_t thread,long line);
int mythreads_destroy(ucontext_t thread_context,int thread_id,long line);
void mythreads_sem_init(sem_t *s,int val,int id);
int mythreads_sem_down(sem_t *s,long line);    int mythreads_sem_up(sem_t *s);
void mythreads_sem_destroy(sem_t *s);
```

```
void mythreads_init(long line) {
    list_init();
    mycoroutines_init(&init_context,line);
    sigemptyset(&signal_mask);
    sigaddset(&signal_mask,SIGALRM);
    act.sa_handler = handler;  act.sa_flags = SA_RESTART;
    sigaction(SIGALRM,&act,NULL);
    timer.it_interval.tv_sec=0;timer.it_interval.tv_usec = 60;
    timer.it_value.tv_sec = 0;    timer.it_value.tv_usec = 60;
    result = list_insert_first(init_context);
    result->sem_id = -1;    result->thread_id = 0;
    mycoroutines_create(&handler_context, NULL,scheduler,NULL,line);
    setitimer(ITIMER_REAL,&timer,NULL);}}
int mythreads_create(ucontext_t thread,void(*body)(),void*arg,long line) {
    mycoroutines_create(&thread,&handler_context, body,arg,line);
    result = list_insert_first(thread);   result->sem_id = -1;
    result->thread_id = counter;    counter++;
    for(current = head->next; current!=head; current=current->next) {}
    return(0);}
void mythreads_sem_init(sem_t *s,int val,int id) {   s->val = val;  s->id = id;}
void mythreads_sem_destroy(sem_t *s) {s = NULL;}
```

```
int mythreads_yield(long line) {
    stored_context = list_search();
    mycoroutines_switch(&(stored_context->context_id),&handler_context,line);
    return(0);}
int mythreads_destroy(ucontext_t context,int thread_id) {
    result = change_state(thread_id,666);
    if(result !=-1){ mycoroutines_switch(&context,&handler_context,line);}
    else {//couldn't locate}   return(0);}
int mythreads_sem_down(sem_t *s,long line) {
  sigprocmask(SIG_BLOCK,&signal_mask,NULL);
  sigwait(&sig_mask, &sig); s->val--;
  if(s->val < 0) {  running_context = list_search();
     change_state(running_context->thread_id,0);  running_context->sem_id = s->id;
     mycoroutines_switch(&(running_context->context_id),&handler_context,line);}
  return(0);}
int mythreads_sem_up(sem_t *s) {
  sigprocmask(SIG_BLOCK,&signal_mask,NULL);
  sigwait(&sig_mask, &sig); s->val++;
  if(s->val <= 0) {
    for(current = head->next; current != head; current = current->next) {
      if((current->state == 0) && (current->sem_id  == s->id)) {
        change_state(current->thread_id,1);  current->sem_id = -1; break}}}return(0);}
```

Άσκηση 4.2: primetest.c

```
extern void handler(int sig) {
  running_context = list_search();
  handler_context.uc_sigmask = signal_mask;
  sigprocmask(SIG_BLOCK,&(handler_context.uc_sigmask),NULL);
  mycoroutines_switch(&(running_context->context_id),
    &handler_context,__LINE__);}
```

```
Main:
 creates/init threads, workers, semaphores
While(i < numbers) {
  mythread_sem_down(mutex);
  If(wait_workers != num_threads)
    {mythread_sem_up(mutex); wait_main++;
  mythread_sem_down(main);
  mythread_sem_down(mutex);}
  for(j < num_threads) {
    if(workers[j].flag == 0 && j < num && I < num) {
      assigns job; flag =-1;
      if(wait_workers != 0) {wait_workers--; up(worker)}}}
mythreads_sem_up(mutex);
}
mythreads_sem_down(main);
for(all workers) → workers[].flag = 666;
mythreads_sem_up(workers);
mythreads_sem_down(main);
destroy all sems;
```

```
Primesearch:
mythread_sem_down(mutex); num_workers++;
If(last worker && wait_main != 0) → wait_main--;
mythreads_sem_up(main);
wait_workers++; mythreads_sem_up(mutex);
mythread_sem_down(workers);
While(1) {
  if(worker.flag != 0) {
    if(terminate) {mythread_sem_down(mutex);
      wait_workers--; mythreads_sem_up(mutex);
      if(wait_works == 0) → mythreads_sem_up(main);
        mythreads_destroy(context_id, thread_id);}
    checks if number prime;}
  else { mythread_sem_down(mutex);
    mythreads_sem_up(workers); mythreads_sem_up(mutex);
    mythread_sem_down(workers); continue;}
  mythread_sem_down(mutex);
  if(wait_main != 0) → wait_main--; mythreads_sem_up(main);
    wait_worker++; worker.flag = 0;
  finished_num++;
  if(all workers blocked && all nums finished) →
    mythreads_sem_up(main);
  mythreads_sem_up(mutex); mythread_sem_down(worker);
}
```