ΤΑΥΤΟΧΡΟΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ 4

Ομάδα 8
Γιαννούκος Τριαντάφυλλος Ανάργυρος
Ματζώρος Χρήστος Κωνσταντίνος

Δηλώσεις και αρχικοποιήσεις μεταβλητών και δομών

```
typedef struct information{
   int value;
   char var_name[NAMELEN];
}table_info;

table_info *GlobalVar;
int GlobalVar_size = 0;

volatile int id = 0;
volatile int num_of_cores = 0;

pthread_mutex_t mtx;
pthread_mutex_t mtx2;

struct list {
   table_info *LocalVar;
   int LocalVar_size;
   FILE *fd;
   int timetostop;
   int is_blocked;
   struct list *next;
   char variable_blocked[LINESIZE];
   char filename[BUFFERSIZE];
   int is_killed;
};

struct list **head;
```

```
void init_list(int id);
void print_list();
struct list *add_node(FILE* input_fd,int id);
struct list *remove_node (struct list *node, int id);
struct list *next_node(struct list *cur_node,int id);
void destroy_list(int id);
int find_core();
int block(char *sem_name, struct list *node_to_block);
int unblock(char *sem_name) ;
int kill_node(int id);
void killed_unblock(struct list *node);
int search_var(table_info *var, int *size, char *name, int is_global)

#define create_new_variable(Var, Var_size, name, position)
#define search_array(.....)


int main(){

head = (struct list **)malloc((sizeof(struct list*))*num_of_cores);

   GlobalVar = (table_info *) malloc(0);

   for(i=0;i<num_of_cores;i++){
      init_list(i);
   }
}
```

Parsing και αποθήκευση μεταβλητών

```
input_fd = fopen(input_filename,"r");
while(1){
   fgets(line_buffer, LINESIZE, cur_node->fd)
   …
   // Search the whole file for LABELS
   …
   token = strtok(line_buffer, delimiter);
   if(strcmp(token, "LOAD")==0){
      // Check for syntax errors
      // Read Instuction Variables
      // Execute Instruction
   }
   else if(strcmp(token, “STORE")==0){
      …
   }
   else if(……) {
      …
}
```

```
#define create_new_variable(Var, Var_size, name, position)
   Var = (table_info *) realloc(Var, sizeof(table_info) * (Var_size+1) );
   strcpy(Var[Var_size].var_name, name);
   position = Var_size;
   Var_size++;

#define search_array(name, name_to_search, pos, pos_string,
        return_pos, Var, Var_size, i, j, name_with_bracket, token, tok)
   if(variable inside brackets) { find variable from array }
   // search for this element of the array
   // if it exists then return the position
   // if the said array doesn’t exist at all then create it
   // if the said array exists but the element that we need does
        not, expand the existing table

int search_var(table_info *var, int *size, char *name, int is_global){
   // go through the variables array
   // if a variable with the given name exists return its position
```

Συγχρονισμός για την κατανομή των νημάτων εφαρμογής

```
int find_core() {
   struct list *current;
   int i = 0;
   int counter = 0;
   int min = 10000;
   int min_id = -1;

   for(i=0;i<num_of_cores;i++){
      // search the list with head[i] to find the
       number of nodes
      if(min>counter){
         min = counter;
         min_id = i;
      }
      counter = 0;
   }
   return min_id;
   // id of the core running the smallest
       amount of applications
}
```

```
void *system_foo(void *arg){
    while(1){
      if(find_core() == core_id){
         // check if data is available in the stdin using select(),
           then read it and create a new app instance
      }
   ….
}
```

Υλοποίηση LOAD και STORE

```
if(strcmp(token, "LOAD")==0){
   pthread_mutex_lock(&mtx);
   token = strtok();  // get local variable
   // check if token is array
   if (variable is an array){
      search_array(...)
   }
   else{
      local_pos =  search_var(cur_node->LocalVar,
                     &cur_node->LocalVar_size, token);
      if(local_pos == -1){
         create_new_variable(cur_node->LocalVar,
                        cur_node->LocalVar_size, token, local_pos)
      }
   }
   token = strtok(); // get global variable
   if (variable is an array){
      search_array(variable is an array)
   }
   else{
      // get original token
      global_pos =  search_var(GlobalVar,&GlobalVar_size, token);
      if(global_pos == -1){
         create_new_variable(GlobalVar, GlobalVar_size, token,
       global_pos)
      }
   }
   cur_node->LocalVar[local_pos].value =
        GlobalVar[global_pos].value;
   pthread_mutex_unlock(&mtx);
}
```

```
else if(strcmp(token, "STORE")==0){
   pthread_mutex_lock(&mtx);
   token = strtok(); // Global Var
   if (variable is an array){
      search_array(variable is an array)
   }
   else{
      // get original token
      global_pos =  search_var(GlobalVar,&GlobalVar_size, token);
      if(global_pos == -1){
         create_new_variable(GlobalVar, GlobalVar_size, token, global_pos)
      }
   }
   token = strtok(); // Local Var
   if(token[0] == '$'){
      if (variable is an array){
         search_array(...)
      }
      else{
         local_pos =  search_var(cur_node->LocalVar,
                        &cur_node->LocalVar_size, token);
         if(local_pos == -1){
            create_new_variable(cur_node->LocalVar,
                           cur_node->LocalVar_size, token, local_pos)
         }
      }
      GlobalVar[global_pos].value = cur_node->LocalVar[local_pos].value;
   }
   else{
      GlobalVar[global_pos].value = atoi(token);
   }
   pthread_mutex_unlock(&mtx);
}
```

Υλοποίηση DOWN και UP

```
if(strcmp(token, "DOWN")==0){
   pthread_mutex_lock(&mtx);
   token = strtok(); // get global variable
   if (variable is an array){
      search_array(variable is an array)
   }
   else{
      global_pos =  search_var(GlobalVar,&GlobalVar_size, token);
      if(global_pos == -1){
         create_new_variable(GlobalVar, GlobalVar_size, token, global_pos)
      }
   }
   if(GlobalVar[global_pos].value == 0){
      block(token, cur_node);
   }
   else{
      GlobalVar[global_pos].value--;
   }
   pthread_mutex_unlock(&mtx);
}

int block(char *sem_name, struct list *node_to_block) {
   struct list *current;
   pthread_mutex_lock(&mtx2);
   for(i=0;i<num_of_cores;i++){
      for (current = head[i]->next; current != head[i]; current = current->next) {
         if((current->is_blocked > 0) && (strcmp(current->variable_blocked,
         sem_name) == 0)){
            if(max_counter < current->is_blocked){
               max_counter = current->is_blocked;
            }
         }
      }
   }
   node_to_block->is_blocked = max_counter + 1;
   strcpy(node_to_block->variable_blocked, sem_name);
   pthread_mutex_unlock(&mtx2);
   return 1;
}
```

```
if(strcmp(token, "UP")==0){
   pthread_mutex_lock(&mtx);
   token = strtok(); // get global variable
   if (variable is an array){
      search_array(variable is an array)
   }
   else{
      global_pos =  search_var(GlobalVar,&GlobalVar_size, token);
      if(global_pos == -1){
         create_new_variable(GlobalVar, GlobalVar_size, token, global_pos)
      }
   }
   if(GlobalVar[global_pos].value == 0){
      res_blocked = unblock(token);
      if(res_blocked == -1){
         GlobalVar[global_pos].value++;
      }
   }
   else{
      GlobalVar[global_pos].value++;
   }
   pthread_mutex_unlock(&mtx);
}

int unblock(char *sem_name) {
   struct list *current;
   pthread_mutex_lock(&mtx);
   for(i=0;i<num_of_cores;i++){
      for (current = head[i]->next; current != head[i]; current = current->next) {
         if((current->is_blocked > 0) && (strcmp(current->variable_blocked, sem_name) == 0)){
            current->is_blocked--;
            counter++;
         }
      }
   }
   pthread_mutex_unlock(&mtx);
   if (counter == 0)
      return -1;
   return counter;
}
```