Prashant Yadav

CS 4348.002 – Program 3

```
/*****************************************************************
 *File: Program3.c
 *Author: Prashant Yadav
 *Procedures:
 *clear_memory - Clear memory locations after each iteration.
 *update_memory_location - Update memory locations to reduce left over time.
 *allocate_static_fixed_mem - Function allocate memory to process statically with equal interval.
 *allocate_static_var_mem - Function allocate memory to process statically with unequal interval.
 *allocate_dynamic_mem - Function allocate memory to process dynamically.
 *reset_time_counter - Reset time counter after each iteration.
 *print_mem_loc - Routine to print the memory locations in order to verify memory locations.
 *complete_queued_process - Routine to update time counter before starting next iteration.
 *allocate_memory - Routine accept a process to allocate memory for given allocation type.
 *simulate_memory_allocation - Routine to simulate memory allocation for 1000 proceses for given
 *                        allocation type.
 *main - Driver main routine from where program starts executing. Here we start simulation for
 *       three different configuration i.e. memory allocation type.
 *****************************************************************/
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
#define TOTAL_PROCESS 1000    //Macro to declare total process
#define SIMULATIONS 1000      //Macro to declare total simulations
#define STATIC_MEM_ALLOC "Static memory allocation with equal block size"  //Static equal sized memory locations macro
#define STATIC_UNEQL_MEM_ALLOC "Static memory allocation with unequal block size"
//Static unequal sized memory locations macro
#define DYNM_MEM_ALLOC "Dynamic memory allocation" //Macro to denote dynamic memory allocation
int static_fixed_mem_alloc[] = {8,8,8,8,8,8,8};  //Block sized memory blocks
int static_variable_mem_alloc[] = {2,4,6,8,8,12,16};  //Unequal sized memory blocks
int memory_location[56];  //56 memory locations
int time_taken=0;  //Track time taken for 1000 processes completion
struct Process{       //Process structure with memory and time requirement as paramter
  int time_required;
  int memory_required;
};

/*****************************************************************
 *void clear_memory()
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: it clears memory locations after each iteration.
 *Parameters:
 *This routine does not take any argument.
 *This routine does not return anything.
 *****************************************************************/
void clear_memory(){
  int i=0;
  for(i=0;i<56;i++){
    memory_location[i]=0;  //Reseting value for each memory location to 0
  }
}
```

```
/*********************************************************************
 *void update_memory_location()
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: It reduces remaining time for each process in memory by 1.
 *Parameters:
 *This routine does not take any argument.
 *This routine does not return anything.
 *********************************************************************/
void update_memory_location(){
  int i=0;
  for(i=0;i<56;i++){
    if(memory_location[i]>0){
      memory_location[i]-=1;    //Reduce value at each memroy location by 1 if it is non zero
    }
  }
}

/*********************************************************************
 *void allocate_static_fixed_mem(int start, int end, struct Process process)
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: It allocates memory for each process with equal sized memory blocks in
 *           memory_location array.
 *Parameters:
 *int start I/P: This gives start index for memory allocation
 *int end I/P: End index of memory location.
 *struct Process process I/P:The process for which memory has to be allocated.
 *This routine does not return anything.
 *********************************************************************/
void allocate_static_fixed_mem(int start, int end, struct Process process){
  int possible_end=7;
  while(possible_end<end){    //Find next possible end for the block.
    possible_end+=8;
  }
  end=possible_end;
  int i;
  for(i=start;i<=end;i++){
    memory_location[i]=process.time_required;
  }
}

/*********************************************************************
 *void reset_time_counter()
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: It resets time counter after each iteration.
 *Parameters:
 *This routine does not take any argument.
 *This routine does not return anything.
 *********************************************************************/
void reset_time_counter(){
  time_taken=0;
}


/*********************************************************************
 *void allocate_static_var_mem(int start, int end, struct Process process)
```

```
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: It allocates memory for each process with unequal sized memory blocks in
 *            memory_location array.
 *Parameters:
 *int start I/P: This gives start index for memory allocation
 *int end I/P: End index of memory location.
 *struct Process process I/P:The process for which memory has to be allocated.
 *This routine does not return anything.
 **********************************************************************/
void allocate_static_var_mem(int start, int end, struct Process process){
  int possible_end=1, i=1;
  while(possible_end<end){    //Find next possible ending for the block.
    possible_end+=static_variable_mem_alloc[i++];
  }
  end = possible_end;
  for(i=start;i<=end;i++){
    memory_location[i]=process.time_required;
  }
}

/**********************************************************************
 *void allocate_dynamic_mem(int start, int end, struct Process process)
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: It allocates memory for each process dynamically in
 *            memory_location array.
 *Parameters:
 *int start I/P: This gives start index for memory allocation
 *int end I/P: End index of memory location.
 *struct Process process I/P:The process for which memory has to be allocated.
 *This routine does not return anything.
 **********************************************************************/
void allocate_dynamic_mem(int start, int end, struct Process process){
  int i;
  for(i=start; i<=end;i++){
    memory_location[i] = process.time_required;
  }
}

/**********************************************************************
 *void print_mem_loc()
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: In order to verify memory allocation this method can be used to print
 *            memory_location array.
 *Parameters:
 *This routine does not take any argument.
 *This routine does not return anything.
 **********************************************************************/
void print_mem_loc(){
  int i;
  for(i=0;i<56;i++){
    printf("%d ",memory_location[i]);
  }
  printf("\n");
}

/**********************************************************************
 *void complete_queued_process()
```

```
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: This routine lets complete all the process before starting next iteration.
 *Parameters:
 *This routine does not take any argument.
 *This routine does not return anything.
 ************************************************************************/
void complete_queued_process(){
  int max_time = memory_location[0],i;
  for(i=1;i<56;i++){
    max_time=max_time<memory_location[i]?memory_location[i]:max_time;
  }
  time_taken+=max_time;     //Update the time_taken variable by max time in location array
  clear_memory();
}


/************************************************************************
 *void allocate_memory(struct Process process, char allocation_type[])
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: This routine allocates memory for given process and for given allocation type.
 *Parameters:
 *struct Process process I/P: Process object for which memory needs to be allocated.
 *char allocation_type[] I/P: Type of memory allocation.
 *int O/P: return 0 if memory is not available otherwise return 1
 ************************************************************************/
int allocate_memory(struct Process process, char allocation_type[]){
  int i,start=0,end=0;
  time_taken++;
  update_memory_location();  //Update memory_locations array before allocating.
  for(i=0;i<56;i++){       //Loop to find first instance where required memory is available.
    if(memory_location[i]==0){
      end++;
      start = memory_location[start]!=0?i:start;
    }
    else{
      end=i;
      start=i;
    }
    if((end-start+1)>=process.memory_required){
        break;
    }
  }
  if((end-start+1)>=process.memory_required){    //If memory is available
    if(strcmp(allocation_type, STATIC_MEM_ALLOC)==0){  //Checks to find type of memory allocation
      allocate_static_fixed_mem(start, end, process);
    }
    else if(strcmp(allocation_type, STATIC_UNEQL_MEM_ALLOC)==0){
      allocate_static_var_mem(start, end, process);
    }else{
      allocate_dynamic_mem(start, end, process);
    }
     return 1;   //return 1 if memory allocation is possible
  }
   return 0;   //return 0 if required memory is not available.
```

```
}


/*********************************************************************
 *void simulate_memory_allocation(char allocation_type[])
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: This routine simulates memory allocation for 1000 process for given
 *           memory allocation type.
 *Parameters:
 *char allocation_type[] I/P: Type of memory allocation.
 *This routine does not return anything.
 *********************************************************************/
void simulate_memory_allocation(char allocation_type[]){
  struct Process process;
  srand(time(0));  //Seed to given random value for each execution
  int i,isAllocated=1,j,simulation_time=0;
    for(j=0;j<SIMULATIONS;j++){
      for(i=0;i<TOTAL_PROCESS;i++){
        if(isAllocated==1){
          process.time_required = 1 + (rand()%10);  //Randomly initialize time requirement
          process.memory_required = 1 + (rand()%15); //Randomly initialize memory requirement
        }
        isAllocated=allocate_memory(process, allocation_type);
      }
      complete_queued_process();
      simulation_time+=time_taken;
      reset_time_counter();
    }
    printf("Time taken for %s is %.2f\n",allocation_type,(float)simulation_time/SIMULATIONS);
}

/*********************************************************************
 *void main()
 *Author: Prashant Yadav
 *Date: 10/13/2019
 *Description: This is driver routine. Program execution starts here. We call
 *           simulate_memory_allocation with all three configurations.
 *Parameters:
 *It does not take any input args.
 *This routine does not return anything.
 *********************************************************************/
void main(){
  simulate_memory_allocation(STATIC_MEM_ALLOC); //Simulate static memory allocation
  simulate_memory_allocation(STATIC_UNEQL_MEM_ALLOC); //Simulate static unequal sized block
memory allocation
  simulate_memory_allocation(DYNM_MEM_ALLOC); //Simulate dynamic memory allocation
}
```