**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

}