Multiprogramming Operating System
(Phase 2)

NAME:Sanket Sudake
CLASS:TE-D BRANCH:COMP BATCH:4
ROLLNO:48

Source:-

```
/*************************************************/
/* File name:-main.c                             */
/* Licence:- GNU GPL v3                          */
/* Project Name:-Multiprogramming  Operating System */
/* *********************************************/

#include"mos.h"
#include"cpu.h"
#include"linep.h"
#include"card.h"
int main(){
  HAL *hal=NULL;
  char instream[]="input.txt";
  char outstream[]="output.txt";
  hal=hal_turnon(hal,instream,outstream);
  /* Read and execute cycle */
  while(1){
    card_read(hal->instream,hal->memory,hal->pcb,hal->cpu);
    mos_execute(hal);
  }
  hal_turnoff(hal);
  return 0;
} /*End main*/
```

```
/*****************************************************/
/* File Name:- card.h                                */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* *************************************************/

#ifndef CARD_H_
#define CARD_H_
#include"mem.h"
#include"pblock.h"
#include"cpu.h"
/* Card functions */
extern FILE* card_open(FILE *FIN,char *FILENAME);
extern int card_read(FILE *FIN,MEM *memory,PCB *pcb,CPU *cpu);
extern void card_close(FILE *FIN);
#endif  /* card.h */
```

```
/*****************************************************/
/* File Name:- card.c                                */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#include"card.h"
#include"mos.h"
/* Open card using filename */
FILE*
card_open(FILE *FIN,char *filename){
  FIN=fopen(filename,"rt");
  if(FIN==0){
    fprintf(stderr,"\ncard_open:Can't Open Input File");
    exit(8);
  }
  return FIN;
}
/* Read card up $DTA and store in buffer */
int
card_read(FILE *FIN,MEM *memory,PCB *pcb,CPU *cpu){
  int count=0,temp=0,r_no=0,dtaflag=0,i=0;
  r_no=mem_random(memory);
  mem_pcount(memory,r_no);
  cpu->PTR=r_no;
  for(i=0;i<42;i++)
    memory->MMEM[r_no][i]='0';
  while(fgets(memory->LINE,(int)sizeof(memory->LINE),FIN)){
    if((*memory).LINE[0]=='$'){
      switch(memory->LINE[1]){
      case 'A':pcb_set((*memory).LINE,pcb);
        break;
      case 'D':temp++;
        dtaflag=1;
        break;
      case 'E':fprintf(stdout,"\ncard_read:Dirty Read :-(");
        return 0;
        break;
      default:fprintf(stdout,"\ncard_read:Case not found for $ statement");
      }
    }
    if(dtaflag)
      break;
    if((*memory).LINE[0]!='$'){
      r_no=mem_random(memory);
      mem_pcount(memory,r_no);
      memory->MMEM[cpu->PTR][count*4+0]=0;
      memory->MMEM[cpu->PTR][count*4+1]=1;
      memory->MMEM[cpu->PTR][count*4+2]=r_no/10;
      memory->MMEM[cpu->PTR][count*4+3]=r_no%10;
      strcpy(&((*memory).MMEM[r_no][0]),(*memory).LINE);
      count++;
```

```
    }
    else{
      temp++;
    }
  }
  switch(temp){
  case 0:fprintf(stdout,"\ncard_read:End of jobs :-)");
    exit(8);
    break;
  case 1:fprintf(stdout,"\ncard_read:Check for $AMJ,$DTA in Program :-(");
    exit(8);
    break;
  case 2://fprintf(stdout,"\ncard_read:Program contains $AMJ,$DTA");
    break;
  default:fprintf(stdout,"\ncard_read:Case not found for temp");
  }
  count=0;
  dtaflag=cpu->PTR;
  fprintf(stdout,"\nPTR=%d\n",cpu->PTR);
  while(memory->MMEM[dtaflag][count*4+1]==1){
    r_no=memory->MMEM[dtaflag][count*4+2]*10+memory->MMEM[dtaflag][count*4+3];
    fprintf(stdout,"%d\t%s",r_no,memory->MMEM[r_no]);
    count++;
  }
  //  fprintf(stdout,"\n");
  return count;
}
/* Close cardreader */
void
card_close(FILE *FIN){
  if(FIN){
    fclose(FIN);
  }
}
```

```
/*****************************************************/
/* File Name:- linep.h                                */
/* Licence:- GNU GPL v3                               */
/* Project Name:-Multiprogramming Operating System    */
/* ***************************************************/
#ifndef LINEP_H_
#define LINEP_H_
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/* LINEPRINTER Structure=>
   OPLINE=Output Line;
   OPFLAG=Check New Output Available,Or Halt
 */
struct LINEPRINTER{
  char OPLINE[42];
  int OPFLAG;
```

```
};
typedef struct LINEPRINTER LINEPRINTER;
/* Functions for lineprinter */
extern LINEPRINTER*
linep_init(LINEPRINTER *linep);
extern FILE*
linep_open(FILE *FOUT,char *filename);
extern void
linep_close(FILE *FOUT);
extern void
linep_print(FILE *FOUT,LINEPRINTER *linep);
extern void
linep_jobend(FILE *FOUT);
#endif  /* linep.h */
```

```
/*****************************************************/
/* File Name:- linep.c                               */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#include"linep.h"
/* Allocate memory for lineprinter */
static LINEPRINTER*
linep_alloc(LINEPRINTER *linep){
  if(linep==NULL){
    linep=(LINEPRINTER*)malloc(sizeof(LINEPRINTER));
  }
  if(linep==NULL){
    fprintf(stderr,"linep_alloc:Memory not allocated");
    exit(8);
  }
  return linep;
}
/* Set all memory locations to null for lineprinter */
static LINEPRINTER*
linep_set(LINEPRINTER *linep){
  memset((void*)(*linep).OPLINE,'\0',sizeof(char)*42);
  return linep;
}
/* Calling innitialize function for linep */
LINEPRINTER*
linep_init(LINEPRINTER *linep){
  linep= linep_alloc(linep);
  linep=linep_set(linep);
  return linep;
}
/* Open file for putting output  */
FILE*
linep_open(FILE *FOUT,char *filename){
  FOUT=fopen(filename,"w+b");
  if(FOUT==NULL){
    fprintf(stderr,"linep_open:Cant open output file");
```

```
    exit(8);
  }
  return FOUT;
}
/* Close output file */
void
linep_close(FILE *FOUT){
  if(FOUT){
      fclose(FOUT);
    }
}
/* Print line output file */
void
linep_print(FILE *FOUT,LINEPRINTER *linep){
  int found=0,i=0;
  if(FOUT && linep->OPFLAG)
    {
      fseek(FOUT,0,SEEK_END);
      /* for(i=0;i<42;i++) */
      /*   { */
      /*     putc(linep->OPLINE[i],FOUT); */
      /*   } */
      for(i=0;i<42;i++)
        if(linep->OPLINE[i]=='\n')
          found=1;
      if(!found)
        linep->OPLINE[strlen(linep->OPLINE)]='\n';
      fputs(linep->OPLINE,FOUT);
    }
  linep->OPFLAG=0;
}
/* End job enter two blank lines */
void
linep_jobend(FILE *FOUT){
  if(FOUT)
    {
      fseek(FOUT,0,SEEK_END);
      fputs("\n\n",FOUT);
    }
}
```

```
/*****************************************************/
/* File Name:- mem.h                                 */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#ifndef MEM_H_
#define MEM_H_
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<memory.h>
```

```
#include<time.h>
/* MEMORY structure=>
   BUFF Buffer Containing 10 words
   MMEM Storing user data
   LINE To Read Line From Program
 */
struct MEM{
  char MMEM[30][42];
  char LINE[50];
  char mtemp[4];
  int randlist[10];
  int pcount;
};
typedef struct MEM MEM;
/* memory functions */
extern MEM*
mem_init(MEM *memory);
extern inline
char meml_getchar(MEM *memory,int no);
extern inline
char memm_getchar(MEM *memory,int line,int place);
extern int
mem_random(MEM *memory);
extern inline
void mem_pcount(MEM *memory,int no);
/* To be implemented */
#endif  /* mem.h */
```

```
/*****************************************************/
/* File Name:- mem.c                                 */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#include"mem.h"
/* Allocate Memory */
static MEM*
mem_alloc(MEM *memory){
  if(memory==NULL)
    memory=(MEM*)malloc(sizeof(MEM));
  if(memory==NULL)
    {
      fprintf(stderr,"mem_alloc:Memory not allocated");
      exit(8);
    }
  return memory;
}
/* Set Whole Memory To Blank */
static MEM*
mem_set(MEM *memory){
  memset((void*)memory->MMEM,'\0',sizeof(char)*30*42);
  memset((void*)memory->LINE,'\0',sizeof(char)*50);
  memset((void*)memory->mtemp,'\0',sizeof(char)*4);
```

```
  memset((void*)memory->randlist,-1,sizeof(int)*10);
  memory->pcount=0;

  return memory;
}
/* Innitialize Memory */
MEM*
mem_init(MEM *memory){
 memory=mem_alloc(memory);
 memory=mem_set(memory);

 return memory;
}
/* Get A Character From Current Memory Line*/
inline char
meml_getchar(MEM *memory,int no){
  return (*memory).LINE[no];
}
/* Get A Character From Main Memory */
inline char
memm_getchar(MEM *memory,int line,int place){
  return (*memory).MMEM[line][place];
}
/* Get Random Address Checked Before Allocation */
static int
mem_randcheck(MEM *memory,int r_no)
{
  int i;
  for (i=0;memory->randlist[i]!=-1;i++)
    if(memory->randlist[i]==r_no)
      return 0;
  return 1;
}
/* Get A Random Number Between 0 to 30 */
int
mem_random(MEM *memory)
{
  int r_no,correct=0;

  while(correct==0){
    r_no=random()%30;
    if(mem_randcheck(memory,r_no))
      {
        correct=1;
        return r_no;
      }
  }
  printf("mem_random:Sorry cant generate random no.\n");
  return -1;
}
/* Update pcount variable for synchronized random no generation */
inline void
```

```
`mem_pcount(MEM *memory,int no){
  memory->randlist[memory->pcount]=no;
  memory->pcount+=1;

}
```

```
/*****************************************************/
/* File Name:- pblock.h                                */
/* Licence:- GNU GPL v3                                */
/* Project Name:-Multiprogramming Operating System     */
/* ***************************************************/

#ifndef PBLOCK_H_
#define PBLOCK_H_
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/* PCB Structure =>
 job_id=Current job id
 TTL=Total time limit
 TLL=Total line limit
*/
struct PCB
{
  int job_id;
  int TTL;
  int TLL;
  char temp[4];
};
typedef struct PCB PCB;
/* Functions for process control block */
extern PCB*
pcb_init(PCB *pcb);
extern void
pcb_set(char *LINE,PCB *pcb);
#endif  /* linep.h */
```

```
/*****************************************************/
/* File Name:- pblock.c                                */
/* Licence:- GNU GPL v3                                */
/* Project Name:-Multiprogramming Operating System     */
/* ***************************************************/
# include"pblock.h"
/* Allocate and innitialize process control block */
PCB*
pcb_init(PCB *pcb)
{
  if(pcb==NULL)
    pcb=(PCB*)malloc(sizeof(PCB));
  if(pcb==NULL)
    fprintf(stdout,"\npcb_init:Memory not allocated");
```

```
  pcb->job_id=0;
  pcb->TTL=0;
  pcb->TLL=0;
  memset((void*)(*pcb).temp,'\0',sizeof(char)*4);
  return pcb;
}
/* Set pcb at start of each process */
void
pcb_set(char *LINE,PCB *pcb)
{
  /*  hal->mem->LINE  Here we get*/
  /* Extract job id */
  pcb->temp[0]=LINE[4];
  pcb->temp[1]=LINE[5];
  pcb->temp[2]=LINE[6];
  pcb->temp[3]=LINE[7];
  pcb->job_id=atoi(pcb->temp);
  /* Extract Total Time limit */
  pcb->temp[0]=LINE[8];
  pcb->temp[1]=LINE[9];
  pcb->temp[2]=LINE[10];
  pcb->temp[3]=LINE[11];
  pcb->TTL=atoi(pcb->temp);
  /* Extract Total Line Limit */
  pcb->temp[0]=LINE[12];
  pcb->temp[1]=LINE[13];
  pcb->temp[2]=LINE[14];
  pcb->temp[3]=LINE[15];
  pcb->TLL=atoi(pcb->temp);
  printf("\npcb_set:JOB_ID=%d\tTTL=%d\tTLL=%d",pcb->job_id,pcb->TTL,pcb->TLL);
}
```

```
/*****************************************************/
/* File Name:- cpu.h                                 */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#ifndef CPU_H_
#define CPU_H_
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
/*CPU Structure=>
  PC=Program Counter,
  TTC=Total Time Counter,
  TLC=Total Line Limit,
  IR=Instruction Register,
  R=CPU Register,
  PTR=Page Table Register,
  SI=System Interrupt,
  PI=Program Interrupt,
  TI=Time Interrupt
```

```
*/
struct CPU{
  unsigned int IC,TTC,TLC,SI,PI,TI,PTR;
  char R[5];
  char IR[5];
  enum MODE{master,slave}MODE;
  enum TOGGLE{t,f}C;
};
typedef struct CPU CPU;
/* Cpu functions */
extern CPU* cpu_init(CPU *cpu);
extern void cpu_em(int n,FILE *FOUT);
#endif  /* cpu.h int addr*/
```

```
/*****************************************************/
/* File Name:- cpu.c                                 */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#include"cpu.h"
/* Innitialize cpu */
CPU*
cpu_init(CPU *cpu){
  if(cpu==NULL)
    cpu=(CPU*)malloc(sizeof(CPU));
  if(cpu==NULL)
    fprintf(stderr,"\ncpu_init:Memory not allocated");
  cpu->IC=0;
  cpu->TTC=0;
  cpu->TLC=0;
  memset((void*)cpu->IR,'\0',sizeof(char)*5);
  memset((void*)cpu->R,'\0',sizeof(char)*5);
  cpu->SI=3;
  cpu->PI=0;
  cpu->TI=0;
  cpu->PTR=0;
  cpu->MODE=slave;
  cpu->C=f;
  return cpu;
}
/* Print error message */
void
cpu_em(int n,FILE *FOUT)
{
  fseek(FOUT,0,SEEK_END);
  switch(n)
    {
    case 0:
      fputs("\ncpu_em:No Error",FOUT);
      break;
    case 1:
      fputs("\ncpu_em:Out of Data",FOUT);
```

```
      break;
    case 2:
      fputs("\ncpu_em:Line Limit Exceeded",FOUT);
      break;
    case 3:
      fputs("\ncpu_em:Time Limit Exceeded",FOUT);
      break;
    case 4:
      fputs("\ncpu_em:Operation Code Error",FOUT);
      break;
    case 5:
      fputs("\ncpu_em:Operand Error",FOUT);
      break;
    case 6:
      fputs("\ncpu_em:Invalid Page Fault",FOUT);
      break;
    default:
      fputs("\ncpu_em:No specified error format",FOUT);
    }
}
```

```
/*******************************************************/
/* File Name:- mos.h                                   */
/* Licence:- GNU GPL v3                                */
/* Project Name:-Multiprogramming Operating System     */
/* ****************************************************/
#ifndef MOS_H_
#define MOS_H_
#include"cpu.h"
#include"pblock.h"
#include"mem.h"
#include"card.h"
#include"linep.h"
/* HAL hardware structure
   cpu-cpu for program execution
   memory-memory for handling all memory related operations
   lineprinter-lineprinter for printing output in output File
   instream-input code tape
   outstream-output tape
*/
struct HAL{
  CPU *cpu;
  MEM *memory;
  LINEPRINTER *linep;
  PCB *pcb;
  FILE *instream;
  FILE *outstream;
  int flag;
};
typedef struct HAL HAL;
/* Turn on & off  HAL to allocate or dealloc memory  */
extern HAL*
```

```
hal_turnon(HAL *hal,char *instream,char *outstream);
extern void
hal_turnoff(HAL *hal);
extern void
mos_execute(HAL *hal);
/* All services */
extern HAL*
mos_init(HAL * hal);
extern void
mos_gd(HAL *hal,int addr);
extern void
mos_pd(HAL *hal,int addr);
extern void
mos_lr(HAL *hal,int addr);
extern void
mos_sr(HAL *hal,int addr);
extern void
mos_cr(HAL *hal,int addr);
extern void
mos_bt(HAL *hal,int addr,int *row,int *line);
extern HAL*
mos_halt(HAL *hal);
#endif  /* mos.h */
```

```
/*****************************************************/
/* File Name:- mos.c                                 */
/* Licence:- GNU GPL v3                              */
/* Project Name:-Multiprogramming Operating System   */
/* ************************************************/
#include"mos.h"
#include<ctype.h>
/* Turn on hardware abstraction layer */
HAL*
hal_turnon(HAL *hal,char *instream,char *outstream){
  if(hal==NULL)
    hal=(HAL*)malloc(sizeof(HAL));
  if(hal==NULL){
    fprintf(hal->outstream,"\nhal_turnon=>Memory not allocated");
  }
  hal->cpu=cpu_init(hal->cpu);
  hal->memory=mem_init(hal->memory);
  hal->linep=linep_init(hal->linep);
  hal->pcb=pcb_init(hal->pcb);
  hal->instream=card_open(hal->instream,instream);
  hal->outstream=linep_open(hal->outstream,outstream);
  srand(time(NULL));
  return hal;
}
/* Free Overall Memory */
void
hal_turnoff(HAL *hal){
  card_close(hal->instream);
```

```
  linep_close(hal->outstream);
  if(hal!=NULL){
    free(hal->cpu);
    free(hal->memory);
    free(hal->linep);
    free(hal->pcb);
    free(hal);
  }
}
/* Innitialize mos in intermediate stages  */
HAL*
mos_init(HAL *hal){
  hal->memory=mem_init(hal->memory);
  hal->cpu=cpu_init(hal->cpu);
  hal->linep=linep_init(hal->linep);
  hal->pcb=pcb_init(hal->pcb);
  hal->flag=0;
  return hal;
}
/* Return random for particular count line no.*/
static int
mos_line(HAL *hal,int count){
  int r_no;
  r_no=hal->memory->MMEM[hal->cpu->PTR][count*4+2]*10+hal->memory->MMEM[hal-
>cpu->PTR][count*4+3];
  return r_no;
}
/* Mos int handling interrupts in MOS*/
void
mos_int(HAL *hal,int addr){
  while(1){
    if(hal->cpu->TI==0 && hal->cpu->SI==1){
      mos_gd(hal,addr);
      break;
    }
    else if(hal->cpu->TI==0 && hal->cpu->SI==2){
      mos_pd(hal,addr);
      break;
    }
    else if(hal->cpu->TI==0 && hal->cpu->SI==3){
      mos_halt(hal);
      cpu_em(0,hal->outstream);
      hal->flag=1;
      break;
    }
    else if(hal->cpu->TI==2 && hal->cpu->SI==1){
      cpu_em(3,hal->outstream);
      hal->flag=1;
      break;
    }
    else if(hal->cpu->TI==2 && hal->cpu->SI==2){
      mos_pd(hal,addr);
```

```
    cpu_em(3,hal->outstream);
    hal->flag=1;
    break;
   }
   else if(hal->cpu->TI==2 && hal->cpu->SI==3){
    cpu_em(0,hal->outstream);
    hal->flag=1;
    break;
   }
   else{
    break;
   }
  }
  while(1){
   if(hal->cpu->TI==0 && hal->cpu->PI==1){
    cpu_em(4,hal->outstream);
    hal->flag=1;
    break;
   }
   else if(hal->cpu->TI==0 && hal->cpu->PI==2){
    cpu_em(5,hal->outstream);
    hal->flag=1;
    break;
   }
   else if(hal->cpu->TI==0 && hal->cpu->PI==3){
    cpu_em(6,hal->outstream);
    hal->flag=1;
    break;
   }
   else if(hal->cpu->TI==2 && hal->cpu->PI==1){
    cpu_em(3,hal->outstream);
    cpu_em(4,hal->outstream);
    hal->flag=1;
    break;
   }
   else if(hal->cpu->TI==2 && hal->cpu->PI==2){
    cpu_em(3,hal->outstream);
    cpu_em(5,hal->outstream);
    hal->flag=1;
    break;
   }
   else if(hal->cpu->TI==2 && hal->cpu->PI==3){
    cpu_em(3,hal->outstream);
    hal->flag=1;
    break;
   }
   else{
    break;
   }
  }
  hal->cpu->SI=0;
  hal->cpu->PI=0;
```

```
  hal->cpu->TI=0;
}
/* Mos call Function called after each instruction loaded in IR
check if opcode and oprand are correct later on call the functions*/
int
mos_call(HAL *hal,int *row,int *line){
  char i1,i2,temp[2]="";
  int addr=0;
  i1=hal->cpu->IR[0];
  i2=hal->cpu->IR[1];
  temp[0]=hal->cpu->IR[2];
  temp[1]=hal->cpu->IR[3];
  if(!isdigit(temp[0]) || !isdigit(temp[1])){
    //cpu_em(5,hal->outstream);
    hal->flag=1;
    hal->cpu->PI=2;
    //    return 0;
  }
  addr=atoi(temp);
  fseek(hal->outstream,0,SEEK_END);
  while(1){
    if(i1=='G' && i2=='D'){
      hal->cpu->SI=1;
      break;
    }
    else if(i1=='P' && i2=='D'){
      hal->cpu->SI=2;
      break;
    }
    else if(i1=='L' && i2=='R'){
      mos_lr(hal,addr);
      break;
    }
    else if(i1=='S' && i2=='R'){
      mos_sr(hal,addr);
      break;
    }
    else if(i1=='C' && i2=='R'){
      mos_cr(hal,addr);
      break;
    }
    else if(i1=='B' && i2== 'T'){
      mos_bt(hal,addr,row,line);
      break;
    }
    else{
      hal->cpu->PI=1;
      break;
    }
  }
  hal->cpu->TTC+=1;
  if(hal->cpu->TTC==hal->pcb->TTL)
```

```
    hal->cpu->TI=2;
  mos_int(hal,addr);
  return 0;
}
/* Mos execute
Execution of each mos code, after loading into memory, starts here.
*/
void
mos_execute(HAL *hal){
  int line=0,row=0,count=0,i;
  hal->cpu->IC=0;
  line=mos_line(hal,count);
  while(1){
    if(hal->cpu->TTC>=hal->pcb->TTL)
      {
       cpu_em(3,hal->outstream);
       break;
      }

    if(row==40){
      count+=1;
      row=0;
      line=mos_line(hal,count);
    }
    hal->cpu->IR[0]=hal->memory->MMEM[line][row];
    hal->cpu->IR[1]=hal->memory->MMEM[line][row+1];
    hal->cpu->IR[2]=hal->memory->MMEM[line][row+2];
    hal->cpu->IR[3]=hal->memory->MMEM[line][row+3];
    //    printf("\nrow=%d\tline=%d\tIR=%s",row,line,hal->cpu->IR);
    row+=4;
    if(hal->cpu->IR[0]=='H'){
      hal->cpu->TTC+=1;
      hal->cpu->IC+=1;
      cpu_em(0,hal->outstream);
      break;
    }
    mos_call(hal,&row,&line);
    //    fprintf(hal->outstream,"%s\t",hal->cpu->IR);
    hal->cpu->IC+=1;
    if(hal->flag)
      break;
  }
  printf("%s",hal->memory->MMEM[hal->cpu->PTR]);
  for(i=0;i<strlen(hal->cpu->IR);i++)
    if(hal->cpu->IR[i]=='\n')
      hal->cpu->IR[i]=' ';
  fseek(hal->outstream,0,SEEK_END);
  fprintf(hal->outstream,"\nmos_execute:JID=%d IC=%d IR=%s TLC=%d\n",hal->pcb-
>job_id,hal->cpu->TTC,hal->cpu->IR,hal->cpu->TLC);
  mos_halt(hal);
}
/* Halt Service -halt the mos by inserting two new lines*/
```

```
HAL*
mos_halt(HAL *hal){
  if(meml_getchar(hal->memory,1)!='E'){
    while(fgets(hal->memory->LINE,(int)sizeof(hal->memory->LINE),hal->instream)){
       if(meml_getchar(hal->memory,0)=='$'&& meml_getchar(hal->memory,1)=='E')
         break;
     }
  }
  linep_jobend(hal->outstream);
  hal=mos_init(hal);
  return hal;
}
/* GD get data service allocate the block for new data ,give error if out of
the data
 */
void
mos_gd(HAL *hal,int addr){
  int temp1,temp2,r_no;
  fgets(hal->memory->LINE,(int)sizeof(hal->memory->LINE),hal->instream);
  temp1=hal->cpu->PTR;
  temp2=(addr/10)*4;
  if(hal->memory->MMEM[temp1][temp2+1]==1){
    r_no=mos_line(hal,addr/10);
  }
  else{
    r_no=mem_random(hal->memory);
    mem_pcount(hal->memory,r_no);
    hal->memory->MMEM[temp1][temp2]=0;
    hal->memory->MMEM[temp1][temp2+1]=1;
    hal->memory->MMEM[temp1][temp2+2]=r_no/10;
    hal->memory->MMEM[temp1][temp2+3]=r_no%10;
  }
  if(meml_getchar(hal->memory,0)=='$' && meml_getchar(hal->memory,1)=='E'){
    cpu_em(1,hal->outstream);
    hal->cpu->SI=3;
    hal->flag=1;
  }
  strncpy(hal->memory->MMEM[r_no],hal->memory->LINE,42);
}
/* PD print data,find random no for page and print data */
void
mos_pd(HAL *hal,int addr){
  int temp1,temp2,r_no;
  temp1=hal->cpu->PTR;
  temp2=addr/10*4;
  if(hal->memory->MMEM[temp1][temp2+1]==1){
    if(hal->cpu->TLC < hal->pcb->TLL){
      r_no=mos_line(hal,addr/10);
      strncpy(hal->linep->OPLINE,hal->memory->MMEM[r_no],42);
      hal->linep->OPFLAG=1;
      linep_print(hal->outstream,hal->linep);
      hal->cpu->TLC+=1;
```

```
    }
    else{
      cpu_em(2,hal->outstream);
      hal->flag=1;
    }
  }
  else{
    cpu_em(6,hal->outstream);
    hal->flag=1;
  }
}
/* LR load register,map particular address and get cpu register loaded with
given location */
void
mos_lr(HAL *hal,int addr){
  int temp1,r_no;
  if(hal->memory->MMEM[hal->cpu->PTR][addr/10*4+1]==1){
    r_no=mos_line(hal,addr/10);
    temp1=addr%10*4;
    hal->cpu->R[0]=hal->memory->MMEM[r_no][temp1+0];
    hal->cpu->R[1]=hal->memory->MMEM[r_no][temp1+1];
    hal->cpu->R[2]=hal->memory->MMEM[r_no][temp1+2];
    hal->cpu->R[3]=hal->memory->MMEM[r_no][temp1+3];
    hal->cpu->R[4]='\0';
  }
  else{
    cpu_em(6,hal->outstream);
    hal->flag=1;
  }
}
/* SR store register,store data in cpu register as per address given */
void
mos_sr(HAL *hal,int addr){
  int temp1,temp2,r_no;
  temp1=hal->cpu->PTR;
  temp2=addr/10*4;
  if(hal->memory->MMEM[temp1][temp2+1]!=1){
    hal->memory->MMEM[temp1][temp2+1]=1;
    r_no=mem_random(hal->memory);
    mem_pcount(hal->memory,r_no);
    hal->memory->MMEM[temp1][temp2+2]=r_no/10;
    hal->memory->MMEM[temp1][temp2+3]=r_no%10;
  }
  r_no=mos_line(hal,addr/10);
  hal->memory->MMEM[r_no][addr%10*4+0]=hal->cpu->R[0];
  hal->memory->MMEM[r_no][addr%10*4+1]=hal->cpu->R[1];
  hal->memory->MMEM[r_no][addr%10*4+2]=hal->cpu->R[2];
  hal->memory->MMEM[r_no][addr%10*4+3]=hal->cpu->R[3];

}
/* CR -compare register, compare data in register with memory location
given */
```

```
void
mos_cr(HAL *hal,int addr){
  int temp1,r_no,i,check_flag=1;
  if(hal->memory->MMEM[hal->cpu->PTR][addr/10*4+1]==1){
    r_no=mos_line(hal,addr/10);
    temp1=addr%10*4;
    for(i=0;i<4;i++)
      if(hal->cpu->R[i]!=hal->memory->MMEM[r_no][temp1+i]){
        check_flag=0;
        break;
      }
    if(check_flag){
      hal->cpu->C=t;
    }
  }
  else{
    hal->flag=1;
    cpu_em(6,hal->outstream);
  }
}
/* Branch toggle if toggle value 1 */
void
mos_bt(HAL *hal,int addr,int *row,int *line){
  int tempaddr=0,i,j,flag=0,temp1,r_no;
  if(hal->cpu->C==t){
    for(i=0;i<9;i++){
      for(j=0;j<10;j++){
        tempaddr++;
        if(tempaddr==addr)
          {
            flag=1;
            break;
          }
      }
      if(flag){
        break;
      }
    }
    temp1=hal->cpu->PTR;
    if(hal->memory->MMEM[temp1][i*4+1]==1){
      r_no=mos_line(hal,i);
    }
    *row=j*4;
    *line=r_no;
  }
}
```

Makefile:-

```
#  File name :- makefile
#  Licence:- GNU GPL v3
#  Project Name:-Machine Operating System
```

```
mos   : card.o cpu.o linep.o main.o mem.o mos.o pblock.o
        gcc -Wall -g -o mos card.o cpu.o linep.o main.o mem.o mos.o pblock.o

card.o:card.c
        gcc -Wall -g -c card.c

cpu.o :cpu.c
        gcc -Wall -g -c cpu.c

linep.o:linep.c
        gcc -Wall -g -c linep.c

main.o:main.c
        gcc -Wall -g -c main.c

mem.o :mem.c
        gcc -Wall -g -c mem.c

mos.o :mos.c
        gcc -Wall -g -c mos.c

pblock.o:pblock.c
        gcc -Wall -g -c pblock.c

clean:
        rm mos *.o *.h.gch

rmback:
        rm *~
```

Input file:-

```
$AMJ020200250004
GD20PD20LR20SR30SR31PD30SR40SR41SR42PD40
SR50SR51PD50SR60PD60H
$DTA
*
$END0202
$AMJ030200130002
GD20GD30LR31SR22LR32SR23PD20SR40PD40H
$DTA
CAT CAN
   EAT RAT
$END0302
$AMJ010200070002
GD20LR36CR20BT06GD30PD30PD20H
$DTA
RAM  IS OLDER THAN  SHRIRAM
NOT IN EXISTANCE
$END0102
$AMJ040100090004
GD20PD20GD30PD30GD40GD50LR20CR30BT10PD40
```

PD50H
$DTA
ABCD
ABCD
DO NOT
MATCH
$END0401
$AMJ150300200010
GD20GD30LR30SR7AGD40LR40SR74GD50LR50
SR75GD60GD80LR80SR71GD90LR90SR72PD70H
$DTA
SHE WENT
 TO
GET
HER
BAG
WE
WORK
$END1503
$AMJ140300500008
GD30LR33SR37GD40LR40SR38LR41SR39PA30
H
$DTA
SHE SELLS SEA SHELLS ON
SHORE
$END1403
$AMJ140300500008
GD30LR33SR37GD40LR40SR38LR41SR39PA30
H
$DTA
$END1403
$AMJ040100090004
GD20PD20GD30PD30GD40GD50LR20CR30BA10PD40
PD50H
$DTA
ABCD
ABCD
DO NOT
MATCH
$END0401
$AMJ040100090004
GD20PD20GD30PD30GD40GD50LR20CR30BT1APD40
PD50H
$DTA
ABCD
ABCD
DO NOT
MATCH
$END0401

Output File:-

```
*
*  *
*  *  *
*  *
```

cpu_em:Line Limit Exceeded
mos_execute:JID=202 IC=15 IR=PD60 TLC=4

CAT CAN EAT RAT
RAT

cpu_em:No Error
mos_execute:JID=302 IC=10 IR=H TLC=2

cpu_em:Invalid Page Fault
mos_execute:JID=102 IC=2 IR=LR36 TLC=0

ABCD
ABCD

cpu_em:Time Limit Exceeded
mos_execute:JID=401 IC=9 IR=BT10 TLC=2

cpu_em:Operand Error
mos_execute:JID=1503 IC=4 IR=SR7A TLC=0

cpu_em:Operation Code Error
mos_execute:JID=1403 IC=9 IR=PA30 TLC=0

cpu_em:Out of Data
mos_execute:JID=1403 IC=1 IR=GD30 TLC=0

ABCD
ABCD

cpu_em:Time Limit Exceeded
cpu_em:Operation Code Error
mos_execute:JID=401 IC=9 IR=BA10 TLC=2

ABCD
ABCD

cpu_em:Time Limit Exceeded
cpu_em:Operand Error
mos_execute:JID=401 IC=9 IR=BT1A TLC=2
