```
BINODE *sched(void)
{
		unsigned int i;
		BINODE *nexttorun =NULL;
		interrupt_disable();
		for(i=O;i<8;i++)
				if(retorun[i].pout_val !=NULL)
				{
						fifo_out(redtorun+i,&nexttorun);
						break;
				}
		interrupt_enable();
		return nexttorun;
}


static FIFO blocked[200];

void sleep(unsigned int sem_index)
{
		unsigned int i;
		BINODE *blockingprocess=NULL;
		blockingprocess=malloc(sizeof(BINODE));
		blockingprocess->info=pindex;
		fifo_in(sem[sem_index].queue,blockingprocess);
		pct[pindex].pstat=1;
		pct[pindex].actualprior=pct[pindex].baseprior;
		save_context(pindex)
		semup(sem_index);
}


void wakeup(int sem_index)
{
unsigned int i;
unsigned int prioridade;
BINODE *proc;
interrupt_disable();
fifo_out(blocked+sem_index,&proc)
for(i=0;i<100;i++)
		if((pct[i].busy)==TRUE && pct[i].pid==proc->info)
		{
		prioridade = pct[i].actualprior;
		fifo_in(redtorun+prioridade,proc);
		sem_up(sem_index);
		pct[i].pstat=2;
		break;
		}
for(i=0;i<100;i++)
		if((pct[i].busy)==TRUE && pindex==pct[i].pid)
		{
		 if(prioridade > pct[i].actualprior)
		 {
				prioritysuspended();
				dispatch;
		 }
		 break;
		}
interrupt_enable();
}


void prioritysеded(void)
{
		unsigned int i;
		unsigned int bonus=0;
		unsigned int prioridade;
		unsigned int pcti;
		BINODE *proc;
		interrupt_disable();
		for(i=0;i<100;i++)
				if(pct[i].busy==TRUE && pindex == pct[i].pid)
				{
						pct[i].actualprior=pct[i].baseprior;
						prioridade=pct[i].actualprior;
						pct[i].pstat=2;
						save_context(i);
						break;
				}
		pcti=i;
		for(i=0;i<prioridade;i++)
				if(redtorun[i].pout_val!=NULL)
				{
						bonus=-1;
						break;
				}
		pct[pcti].actualprior;
		proc=malloc(sizeof(BINODE));
		proc->info=pindex;
		fifo_in(redtorun+prioridade,proc);
		interrupt_enable();
		}
}


void dispatch(void)
{
		BINODE *proc=NULL;
		unsigned int i;
		interrupt_disable();
		proc=sched();
		if(proc!=NULL)
		{
		 for(i=O;i<100;i++)
				if(pct[i].busy == TRUE && pct[i].pid == proc -> info)
				{
						pct[i].pstat=o;
						restore_context(i);
						break;
				}
		 free(proc);
		}
		interrupt_enable();
}









void timerrunout(void)
{
		BINODE *n;
		unsigned int i,   k=40;
		BOOLEAN recreditar;

		interrupt_disable();
		save_context(pindex);
		pct[pindex].pstat=2; //READY_TO_RUN
		n=malloc(sizeof(BINODE));
		n.info=pindex;
		fifo_in(redtorun+pct[pindex].prior,n);
		i=1;
		recreditar=TRUE;
		do
		{
				if(!fifo_empty(/*   */))
						recreditar=FALSE;
				i++;
		}while((i<k)&& (recreditar==TRUE));
		if(recreditar==TRUE)
				recredit();
		interrupt_enable();

		}
}


void fifoIN (FIFO *f, char car)
{
		semdown (man.access);
		f->mem[f->in]=car;
		f->in=(f->in+1)%k;
		semup(man.access);
}

typedef struct
{
		unsigned int in,out;
		char mem[k];
		BOOLEAN full;
}


/*void fifoIN (FIFO *f, char car)
{
		f->mem[f->in]=car;
		f->in=(f->in+1)%k;
		if(f->in==f->out)
				f->full=TRUE;
}/*


BOOLEAN fifoinit (FIFO *f)
{
		return		f->full;
}


void fifofull (FIFO *f)
{
		f->in=0;
		f->out=0;
		f->full=FALSE;
}


BOOLEAN fifoEmpty (FIFO *f)
{
		if(f->in==f->out)
				return		f->full;
		else
				return FALSE;
}


int allocBuffer (void)
{
		int i;
		semdown(man.access);
		for (i=0;i<ML;i++)
				if(man.buffer_map[i]==0)
				{
						man.buffer_map[i]=1;
						break;
				}
		man.nbuff++;
		semUp(man.access);
		return i;
}


/*int allocBuffer (void)
{
		int p,i,n;
		semDown(man.access);
		if(man.nt_buffer== man.n_buff)
		{
				semUp(man.access);
				return -1;

		}

		for (i=0;i<ML;i++)
		{
				n=1<<(i%sizeof(int))
				p=i/sizeof(int);
				if((man.buff_map[p] & n)==0)
				{
						man.buff.map[p] = man.buff.map[p] | n;
						man.n_buff++;
						return n;
				}
		}
		semUp(man.access);
		return -1;
		}
}*/




void freeBuffer(int buffId)
{
		unsigned int i,n;
		i=buffId/sizeof(int);
		n=1<<(buffId%sizeof(int));
		semDown(man.access);
		if((man.buff_map[i] & n) == n)
		{
				man.buff_map[i]=man.buff_map[i] & (~n);
				man.n_buff--;
		}
		semUp(man.access);
}
```

```
int getPortId(void)
{
        unsigned int d;
        for(d=0 ; d<8 ;d++)
        {
                semDown(env[d].access);
                if(env[d].sess[env[d].sess_act].pid == getpid())
                {
                        semUp(env[d].access);
                        return d;
                }
                semUp(env[d].access);
        }
        return -1;
}


int getBuffId(void)
{
        int d,aux;
        if((d=getportId()) !=1)
        {
                semDown(env[d].access);
                aux=env[d].sess[env[d].sess_act].combid;
                semUp(env[d].access);
                return aux;
        }
        else
                return -1;
}

void create_session(void)
{
        int i;
        int p=getportId();
        semDown(env[p].access);
        for(i=o;i<10;i++)
                if(sess[i].combid==-1)
                {
                        sess[i].combid=allocBuffer();
                        sess[i].pid=getpid();
                }
        semUp(env[p].access);
}

/*void create_session(void)
{
        int p,b,n=0;
        p=getportId();
        if((p!=-1) && ((b=allocBuffer()) != -1))
        {
                semDown(env[p].access);
                while(n<10)
                {

        if(env[p].sess[(env[p].sess_act+n)%10].combid==-1)
                                {

        env[p].sess[(env[p].sess_act+n)%10].combid=b;

        env[p].sess[(env[p].sess_act+n)%10].combid=getpid();
                                                        break;
                                }
                                n++;
                }
        semUp(env[p].access);
        }
}*/

void free_session(void)
{
        int p,s_a;
        p=getportId();
        semDown(env[p].access);
        s_a=env[p].sess_act;
        semUp(env[p].access);
        NextSession();
        semDown(env[p].access);
        if(s_a != env[p].sess_act)
        {
                freebuffer(env[p].sess[s_a].combid);
                env[p].sess[s_a].combid=-1;
        }
        semUp(env[p].access);
}

void NextSession(void)
{
        int p,i,n;
        p=getportId();
        semDown(env[p].access);
        i=(env[p].sess_act+1)%10;
        n=0;
        while(n<10)
        {
                if(env[p].sess[i].combid != -1)
                        break;
                n++;
                i=(i+1 %10);
        }
        env[p].sess_act=i;
        semUp(env[p].access);
}



void writeNBytes(int n, char buff[])
{
        int aux,i;
        if((aux=getBuffTd() != -1))
                for(i=0;i<n;i++)
                {
                        semDown(buff[aux].fo_full);
                        fifoIn(&buff[aux].f_out,&buff[i]);
                }
}


void readNBytes(int n, char buff[])
{
        unsigned int aux,i;
        aux=getBuffTd();
                for(i=0;i<n;i++)
                {
                        semDown(buff[aux].fi_empty);
                        fifoOut(&buff[aux].f_in,&buff[i]);
                }
}
```

```
void sem_down (unsigned int sem_index)
{
  interrupt_disable();
  if (sem[sem_index].val==0)
    sleep(sem_index);
  else sem[sem_index].val-=1;
 interrupt_enable();
 }


void sem_up (unsigned int sem_index)
{
  BINODE *p;
  interrupt_disable();
  if ([sem_index].val==NULL)
    wakeup(sem_index);
  else sem[sem_index]. val+=1;
  interrupt_enable();
}



BOOLEAN RTCservice(void)
{
if ((pindex >=100)  && (!pct[pindex].busy) && (pct[pindex].pstart !=RUN))
                return FALSE;

if(pct[pindex].class== SCHED_OTHER)
{
                if (pct[pindex].prior!=0)
                        pct[pindex].prior --;
                if (pct[pindex].prior==0)
                        return TRUE;
                }
                return FALSE;
}
}


/*algoritmo do relógio – Pseudo Código*/
while (! frame_adequado)
{
    if(Ref= =0)
        substituicao(pag_associada);
   else
        Ref=0;
        ponteiro + +     % nº elementos
}


BINODE *getprocess (void)
{ BINODE *val=NULL; int i;
BOOLEAN sff=FALSE;
  for (i=o;i<2;i++)
     {
                if(! fifo_empty(& sff_rtr[i]))
                {
                        ssf=TRUE;
                        fifo_out(& sff_rtr[i],&val);
                        break;
            }
   }
   if(!sff)
   {
     for (i=39;i>0;i--)
       {
                if(! fifo_empty(& soth_rtr[i]))
                {          fifo_out(& soth_rtr[i],&val);
                        break; }
       }
    }
return(&val);
}


void prioritysuperseded(void)
{
BINODE *val;
if((pindex  <100)      &&  (pct[pindex].busy)   &&   (pct[pindex].pstart   =   =RUN)&&(
pct[pindex].pior<40))
{               if(pct[pindex].class = =SCHED_OTHER)
                { save_context();
                  pct[pindex].pstart =READ_TO_RUN;
                  val=malloc(sizeof(BINODE));
                  val.info=pindex;
                  fifo_in(&soth_rtr [pct[pindex].prior],&val);
                }
else if((pct[pindex].class = =SCHED_FIFO)&& (pct[pindex].pior= =1))
 {              save_context();
                pct[pindex].pstart =READ_TO_RUN;
                val=malloc(sizeof(BINODE));
                val.info=pindex;
                fifo_in(&sff_rtr [1],&val);
 }
}
}


void recredit(void)
{
     BINODE *val;
     int i;
  for (i=o;i<100;i++)
    {
      If((pct[i].busy) && (pct[i].class= =SCHED_OTHER)
        &&( (pct[i].pstart==BLOCKED))
       {
         pct[i].pior=pct[i].pior/2+ pbase[i];
       }
      else if (busy,class,rtr,fifo_empty[0])
       }
      pct[i].pior=pbase[i];
      fifo_out(&soth_rtr[0],&val);
      fifo_in(&soth_rtr[pct[i].pior],&val);

}

void fifoOUT(FIFO *f, char *carp)
{
  If(!(f->full) && (f->ii == f->ir))
  {
          *carp = NULL;
          return;
  }

  *carp = f-> mem[f->fir];
  f->ir +=1;
  f->ir % = k;
  f->full = FALSE;

}
```