**Contiguous file allocations**

#include <stdio.h>

#include <string.h>

int num=0, length[10], start[10];

char fid[20][4], a[20][4];

void directory()

{

int i;

printf("\nFile Start Length\n");

for(i=0; i<num; i++)

printf("%-4s %3d %6d\n",fid[i],start[i],length[i]);

}

void display()

{

int i;

for(i=0; i<20; i++)

printf("%4d",i);

printf("\n");

for(i=0; i<20; i++)

printf("%4s", a[i]);

}

main()

{

int i,st,nb,ch,flag;

char id[4];

for(i=0; i<20; i++)

strcpy(a[i], "");

printf("Disk space before allocation:\n");

display();

do

{

printf("\nEnter File name (max 3 char) : ");

scanf("%s", id);

printf("Enter start block : ");

scanf("%d", &st);

printf("Enter no. of blocks : ");

scanf("%d", &nb);

strcpy(fid[num], id);

length[num] = nb;

flag = 0;

if((st+nb) > 20)

{

printf("Requirement exceeds range\n");

continue;

}

for(i=st; i<(st+nb); i++)

if(strcmp(a[i], "") != 0)

flag = 1;

if(flag == 1)

{

printf("Contiguous allocation not possible.\n");

continue;

}

start[num] = st;

for(i=st; i<(st+nb); i++)

strcpy(a[i], id);;

printf("Allocation done\n");

num++;

printf("\nAny more allocation (1. yes / 2. no)? : ");

scanf("%d", &ch);

} while (ch == 1);

printf("\n\t\t\tContiguous Allocation\n");

printf("Directory:");

directory();

printf("\nDisk space after allocation:\n");

display();

printf("\n");

}

**Producer-consumer program**

#include<stdio.h>

#include <stdlib.h>

int mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

**Producer\_Consumer(or u can follow this also)**

#include<stdio.h>

#include<stdlib.h>

int temp = 0, overflow = 0, mutex = 1, underflow = 10;

int wait\_protocol(int counter)

{

return (--counter);

}

int signal\_protocol(int counter)

{

return(++counter);

}

void producer()

{

mutex = wait\_protocol(mutex);

     overflow = signal\_protocol(overflow);

     underflow = wait\_protocol(underflow);

     temp++;

     printf("\nProducer:\t Element %d\n", temp);

     mutex = signal\_protocol(mutex);

}

void consumer()

{

     mutex = wait\_protocol(mutex);

     overflow = wait\_protocol(overflow);

     underflow = signal\_protocol(underflow);

     printf("\nConsumer:\t Element %d\n", temp);

     temp--;

     mutex = signal\_protocol(mutex);

}

int main()

{

int choice;

     printf("\n1.Producer\t");

printf("2. Consumer\t");

printf("3. Quit\t");

     while(1)

     {

         printf("\nEnter your choice:\t");

         scanf("%d", &choice);

         switch(choice)

         {

             case 1: if((mutex == 1) && (underflow != 0))

                         {

producer();

}

                     else

{

                         printf("Buffer Overflow\n");

}

                     break;

             case 2: if((mutex == 1) && (overflow != 0))

                         {

consumer();

}

                     else

{

                         printf("Buffer Underflow\n");

}

                     break;

             case 3: exit(0);

                     break;

         }

     }

     return 0;

}

**Pipes(follow who command program itself)**

To determine number of users logged in using pipe.

Procedure:

1. Decalre a array pfd with two elements for pipe descriptors.

2. Create pipe on pfd using pipe function call.

a. If return value is -1 then stop

3. Using fork system call, create a child process.

4. Free the standard output (1) using close system call to redirect the output to pipe.

5. Make a copy of write end of the pipe using dup system call.

6. Execute who command using execlp system call.

7. Free the standard input (0) using close system call in the other process.

8. Make a close of read end of the pipe using dup system call.

9. Execute wc –l command using execlp system call.

10. Stop

Thus standard output of who is connected to standard input of wc using pipe to compute

number of users logged in.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

int pfds[2];

pipe(pfds);

if (!fork())

{

close(1);

dup(pfds[1]);

close(pfds[0]);

execlp("who", "who", NULL);

}

else

{

close(0);

dup(pfds[0]);

close(pfds[1]);

execlp("wc", "wc", "-l", NULL);

}

}

**NOTE: for round robin u have to consider arrival time also and for scheduling programs drawing gantt chart is necessary….**