Lab4b:

Code:

#include<sys/ipc.h>

#include<stdio.h>

#include<string.h>

#include<sys/msg.h>

#include<stdlib.h>

#include <unistd.h>

struct

{

long mtype;

char mtext[20];

}send,recv;

int main()

{

int qid,pid,len;

qid=msgget((key\_t)0X2000,IPC\_CREAT|0666);

if(qid==-1)

{

perror("\n message failed");

exit(1);

}

send.mtype=1;

strcpy(send.mtext,"\n hello i am parent");

len=strlen(send.mtext);

pid=fork();

if(pid>0)

{

if(msgsnd(qid,&send,len,0)==-1)

{

perror("\n message sending failed");

exit(1);

}

printf("\n message has been posted");

sleep(2);

if(msgrcv(qid,&recv,100,2,0)==-1)

{

perror("\n msgrcv error:");

exit(1);

}

printf("\n message received from child - %s\n",recv.mtext);

}

else

{

send.mtype=2;

strcpy(send.mtext,"\n hi i am child");

len=strlen(send.mtext);

if(msgrcv(qid,&recv,100,1,0)==-1)

{

perror("\n child message received failed");

exit(1);

}

if(msgsnd(qid,&send,len,0)==-1)

{

perror("\n child message send failed");

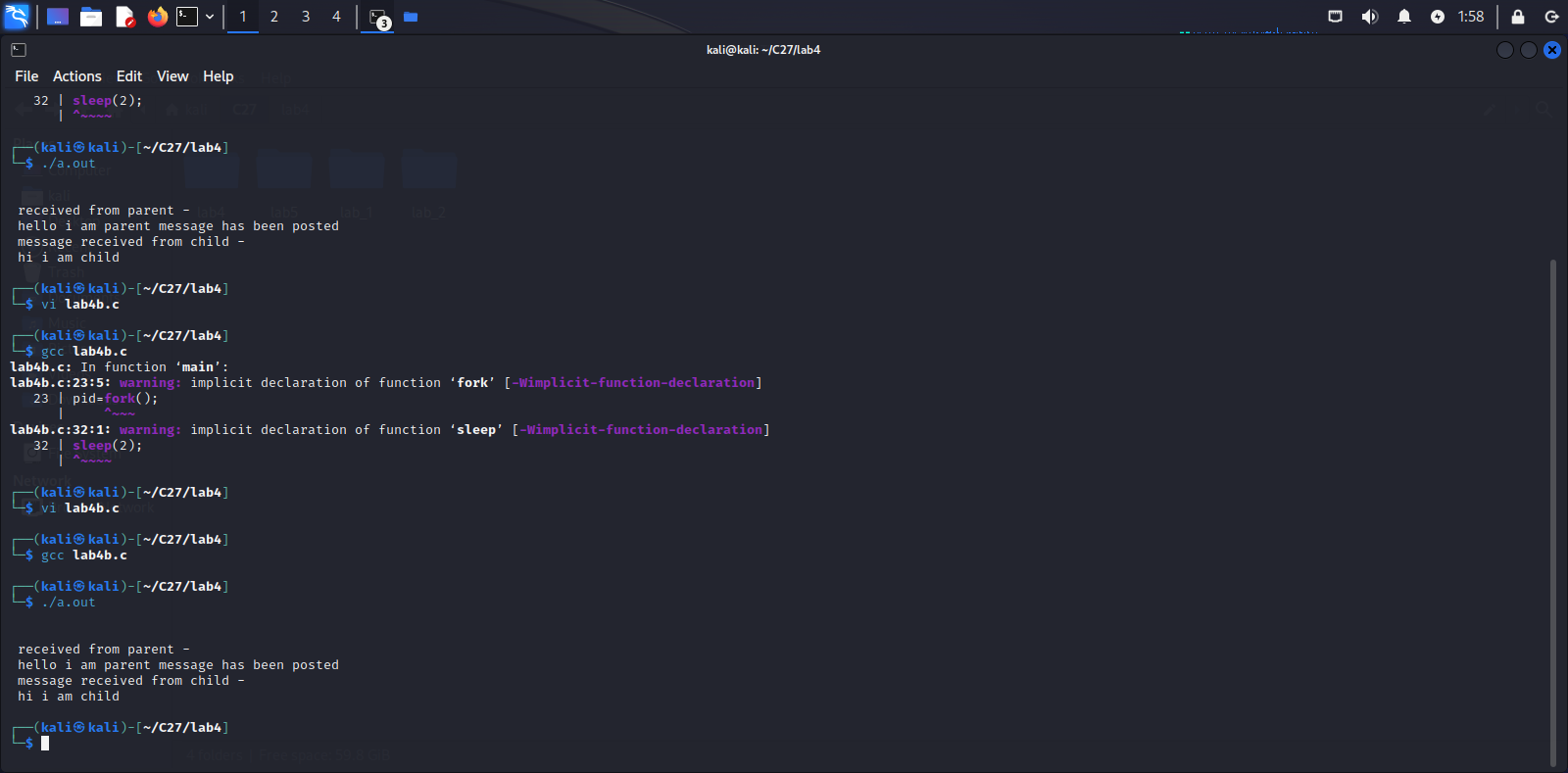
}

printf("\n received from parent - %s",recv.mtext);

}

}

Output:



Lab5a.c:

Code:

#include<sys/ipc.h>

#include<stdio.h>

#include<string.h>

#include<sys/msg.h>

#include<stdlib.h>

int one();

int two();

int tph, philname[20], status[20], howhung, hu[20], cho;

int main()

{

int i;

//clrscr();

printf("\n\nDINING PHILOSOPHER PROBLEM");

printf("\nEnter the total no. of philosophers: ");

scanf("%d",&tph);

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

{

philname[i] = (i+1);

status[i]=1;

}

printf("How many are hungry : ");

scanf("%d", &howhung);

if(howhung==tph)

{

printf("\nAll are hungry..\nDead lock stage will occur");

printf("\nExiting..");

}

else

{

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

{

printf("Enter philosopher %d position: ",(i+1));

scanf("%d", &hu[i]);

status[hu[i]]=2;

}

do

{

printf("1.One can eat at a time\t2.Two can eat at a time\t3.Exit\nEnter your choice:");

scanf("%d", &cho);

switch (cho)

{

case 1: one();

break;

case 2: two();

break;

case 3: exit(0);

default: printf("\nInvalid option..");

}

}

while(1);

}

}

int one()

{

int pos=0, x, i;

printf("\nAllow one philosopher to eat at any time\n");

for(i=0;i<howhung; i++, pos++)

{

printf("\nP %d is granted to eat", philname[hu[pos]]);

for(x=pos;x<howhung;x++)

printf("\nP %d is waiting", philname[hu[x]]);

}

}

int two()

{

int i, j, s=0, t, r, x;

printf("\n Allow two philosophers to eat at same time\n");

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

{

for(j=i+1;j<howhung;j++)

{

if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)

{

printf("\n\ncombination %d \n", (s+1));

t=hu[i];

r=hu[j];

s++;

printf("\nP %d and P %d are granted to eat", philname[hu[i]],

philname[hu[j]]);

for(x=0;x<howhung;x++)

{

if((hu[x]!=t)&&(hu[x]!=r))

printf("\nP %d is waiting", philname[hu[x]]);

}

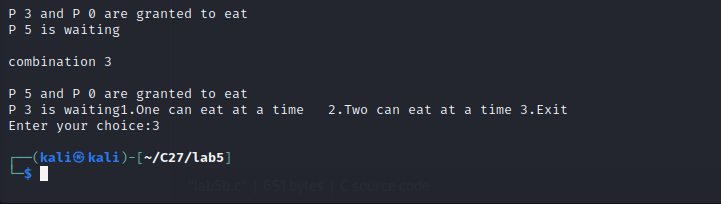
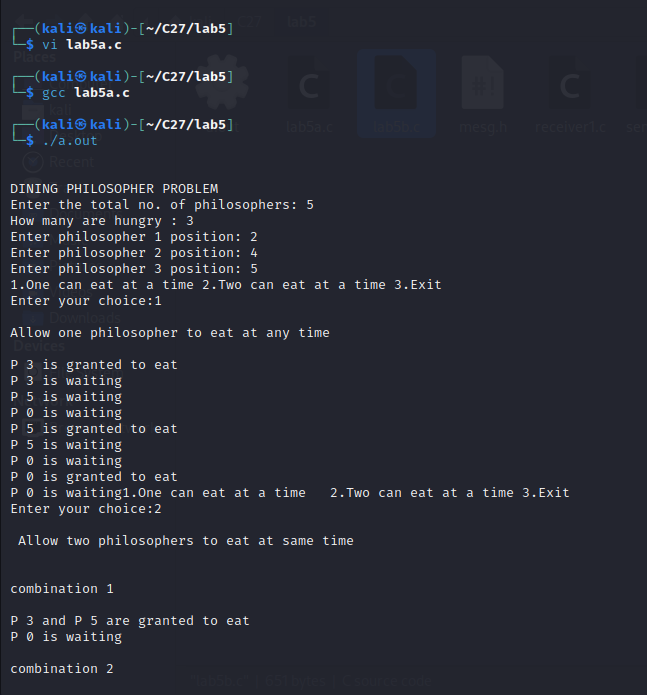
}

}

}

}

Ouptut:



Lab5b.c

Code:

#include<stdio.h>

void main()

{

int buffer[10], bufsize, in, out, produce, consume, choice=0;

in = 0;

out = 0;

bufsize = 10;

while (choice !=3)

{

printf("\n1. Produce \t 2. Consume \t3. Exit");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1: if((in+1)%bufsize==out)

printf("\nBuffer is Full");

else

{

printf("\nEnter the value: ");

scanf("%d", &produce);

buffer[in] = produce;

in =((in+1)%bufsize);

}

break;

case 2: if(in == out)

printf("\nBuffer is Empty");

else

{

consume = buffer[out];

printf("\nThe consumed value is %d", consume);

out = (out+1)%bufsize;

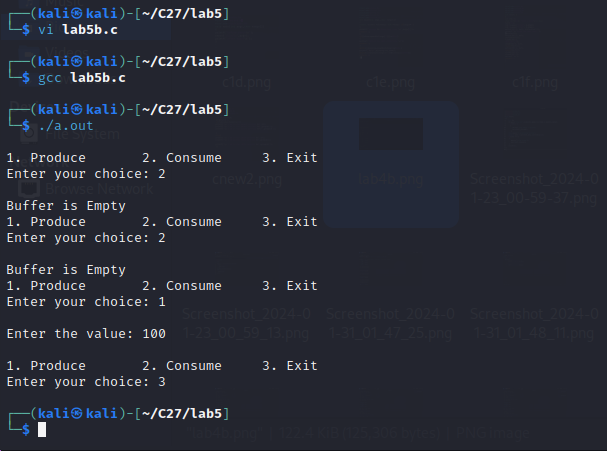
} break;

}

}

}

Output:



Lab5c.c

Code:

Msg.h

//File: mesg.h

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdio.h>

#include<stdlib.h>

#define MKEY1 5543L

#define MKEY2 4354L

#define PERMS 0666

typedef struct

{

long mtype;

char mdata[50];

}mesg;

**Sender1.c:**

//File: sender1.c

#include "mesg.h"

#include<fcntl.h>

#include <errno.h>

#include <fcntl.h>

#include<unistd.h>

#include<stdio.h>

mesg msg;

int main()

{

int mq\_id;

int n;

if((mq\_id=msgget(MKEY1,PERMS|IPC\_CREAT))<0)

{

printf("Sender: Error creating message");

exit(1);

}

msg.mtype=1111L;

n=read(0,msg.mdata,50);

msg.mdata[n]='\0';

msgsnd(mq\_id,&msg,50,0);

}

**receiver1.c**:

#include "mesg.h"

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

mesg msg;

int main()

{

int mq\_id;

int n;

if( ( mq\_id=msgget(MKEY1, PERMS|IPC\_CREAT ) ) < 0)

{

printf("receiver: Error opening message");

exit(1);

}

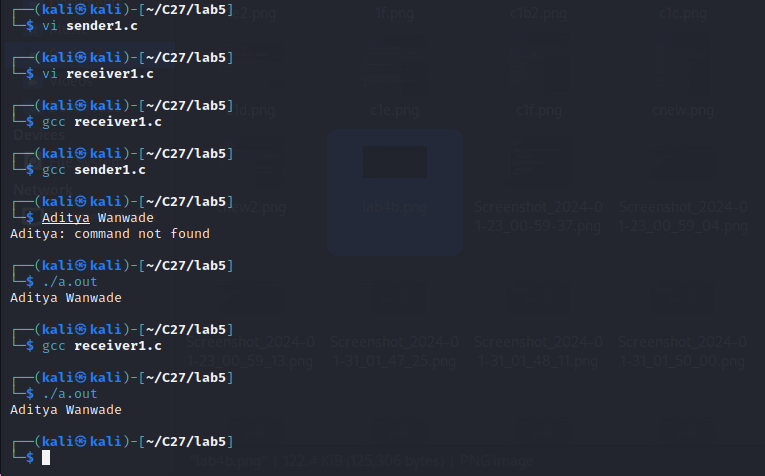
msgrcv(mq\_id,&msg,50,1111L,0);

write(1,msg.mdata,50);

msgctl(mq\_id,IPC\_RMID,NULL);

}

Output:



6 code:

/\* Bankers algorithm for Deadlock Avoidance \*/

#include <stdio.h>

struct process

{

    int allocation[3];

    int max[3];

    int need[3];

    int finish;

} p[10];

int main()

{

    int n, i, I, j, avail[3], work[3], flag, count = 0, sequence[10], k = 0;

    printf("\nEnter the number of process:");

    scanf("%d", &n);

    for (i = 0; i < n; i++)

    {

        printf("\nEnter the %dth process allocated resources:", i);

        scanf("%d%d%d", &p[i].allocation[0], &p[i].allocation[1], &p[i].allocation[2]);

        printf("\nEnter the %dth process maximum resources:", i);

        scanf("%d%d%d", &p[i].max[0], &p[i].max[1], &p[i].max[2]);

        p[i].finish = 0;

        p[i].need[0] = p[i].max[0] - p[i].allocation[0];

        p[i].need[1] = p[i].max[1] - p[i].allocation[1];

        p[i].need[2] = p[i].max[2] - p[i].allocation[2];

    }

    printf("\nEnter the available vector:");

    scanf("%d%d%d", &avail[0], &avail[1], &avail[2]);

    for (i = 0; i < 3; i++)

        work[i] = avail[i];

    while (count != n)

    {

        count = 0;

        for (i = 0; i < n; i++)

        {

            flag = 1;

            if (p[i].finish == 0)

                if (p[i].need[0] <= work[0])

                    if (p[i].need[1] <= work[1])

                        if (p[i].need[2] <= work[2])

                        {

                            for (j = 0; j < 3; j++)

                                work[j] += p[i].allocation[j];

                            p[i].finish = 1;

                            sequence[k++] = i;

                            flag = 0;

                        }

            if (flag == 1)

                count++;

        }

    }

    count = 0;

    for (i = 0; i < n; i++)

        if (p[i].finish == 1)

            count++;

    printf("\n The safe sequence is:\t");

    if (count++ == n)

        for (i = 0; i < k; i++)

            printf("%d\n", sequence[i]);

    else

        printf("SYSTEM IS NOT IN A SAFE STATE \n\n");

    return 0;

}

Output:

A screenshot of a computer

Description automatically generated

7 code:  
#include <stdio.h>

void main()

{

    int found, flag, l, p[4][5], tp, c[4][5], i, j, k = 1, m[5], r[5], a[5], temp[5], sum = 0;

    printf("enter total no of processes: \n");

    scanf("%d", &tp);

    printf("enter clain matrix: \n");

    for (i = 0; i < 4; i++)

    {

        for (j = 0; j < 5; j++)

        {

            scanf("%d", &c[i][j]);

        }

    }

    printf("enter allocation matrix: \n");

    for (i = 0; i < 4; i++)

    {

        for (j = 0; j < 5; j++)

        {

            scanf("%d", &p[i][j]);

        }

    }

    printf("enter resource vector: \n");

    for (i = 0; i < 5; i++)

    {

        scanf("%d", &r[i]);

    }

    printf("enter availability vector: \n");

    for (i = 0; i < 5; i++)

    {

        scanf("%d", &a[i]);

        temp[i] = a[i];

    }

    for (i = 0; i < 4; i++)

    {

        sum = 0;

        for (j = 0; j < 5; j++)

        {

            sum += p[i][j];

        }

        if (sum == 0)

        {

            m[k] = i;

            k++;

        }

    }

    for (i = 0; i < 4; i++)

    {

        for (l = 1; l < k; l++)

        {

            if (i != m[l])

            {

                flag = 1;

                for (j = 0; j < 5; j++)

                {

                    if (c[i][j] > temp[j])

                    {

                        flag = 0;

                        break;

                    }

                }

            }

        }

        if (flag == 1)

        {

            m[k] = i;

            k++;

            for (j = 0; j < 5; j++)

                temp[j] += p[i][j];

        }

    }

    printf("deadlock causing processes are: \n");

    for (j = 0; j < tp; j++)

    {

        found = 0;

        for (i = 1; i < k; i++)

        {

            if (j == m[i])

                found = 1;

        }

        if (found == 0)

            printf("%d\t", j);

    }

}

Output:

A black screen with text and numbers

Description automatically generated

8a code:

/\* A program to simulate FIFO Page Replacement Algorithm \*/

#include <stdio.h>

int main()

{

    int a[5], b[20], n, p = 0, q = 0, m = 0, h, k, i, q1 = 1;

    char f = 'F';

    printf("Enter the Number of Pages:");

    scanf("%d", &n);

    printf("Enter %d Page Numbers:", n);

    for (i = 0; i < n; i++)

        scanf("%d", &b[i]);

    for (i = 0; i < n; i++)

    {

        if (p == 0)

        {

            if (q >= 3)

                q = 0;

            a[q] = b[i];

            q++;

            if (q1 < 3)

            {

                q1 = q;

            }

        }

        printf("\n%d", b[i]);

        printf("\t");

        for (h = 0; h < q1; h++)

            printf("%d", a[h]);

        if ((p == 0) && (q <= 3))

        {

            printf("-->%c", f);

            m++;

        }

        p = 0;

        for (k = 0; k < q1; k++)

        {

            if (b[i + 1] == a[k])

                p = 1;

        }

    }

    printf("\nNo of faults:%d", m);

}

Output:

A black screen with a black background

Description automatically generated

8b code:

/\* A program to simulate LRU Page Replacement Algorithm \*/

#include <stdio.h>

int main()

{

    int a[5], b[20], p = 0, q = 0, m = 0, h, k, i, q1 = 1, j, u, n;

    char f = 'F';

    printf("Enter the number of pages:");

    scanf("%d", &n);

    printf("Enter %d Page Numbers:", n);

    for (i = 0; i < n; i++)

        scanf("%d", &b[i]);

    for (i = 0; i < n; i++)

    {

        if (p == 0)

        {

            if (q >= 3)

                q = 0;

            a[q] = b[i];

            q++;

            if (q1 < 3)

            {

                q1 = q;

            }

        }

        printf("\n%d", b[i]);

        printf("\t");

        for (h = 0; h < q1; h++)

            printf("%d", a[h]);

        if ((p == 0) && (q <= 3))

        {

            printf("-->%c", f);

            m++;

        }

        p = 0;

        if (q1 == 3)

        {

            for (k = 0; k < q1; k++)

            {

                if (b[i + 1] == a[k])

                    p = 1;

            }

            for (j = 0; j < q1; j++)

            {

                u = 0;

                k = i;

                while (k >= (i - 1) && (k >= 0))

                {

                    if (b[k] == a[j])

                        u++;

                    k--;

                }

                if (u == 0)

                    q = j;

            }

        }

        else

        {

            for (k = 0; k < q; k++)

            {

                if (b[i + 1] == a[k])

                    p = 1;

            }

        }

    }

    printf("\nNo of faults:%d", m);

}

Output:

A black screen with text

Description automatically generated