Deadlock Avoidance with Modified Bankers Algorithm to handle interrupts

CSE-2005
Operating Systems
(J Component)
Final Project Report

Project Submitted by:

18BCE0715 – Sanjit C K S

Under the guidance of Dr. Santhi H
Submitted in Partial Fulfilment of the degree of
B.tech in
Computer Science and Engineering, School of Computer Science and Engineering

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Abstract:

An interrupt is an input signal to the processor indicating an event that needs immediate attentionInterrupts alert the processor to a high priority process requiring interruption of the current working process. In I/O devices the Interrupt Service Routine (ISR) is dedicated for this purpose.When a device raises an interrupt at lets say process i, the processor first completes the execution of instruction i. Then it loads the Program Counter (PC) with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process i+1.While the processor is handling the interrupts, it must inform the device that its request has been recognized so that it stops sending the interrupt request signal. Also, saving the registers so that the interrupted process can be restored in the future, increases the delay between the time an interrupt is received and the start of the execution of the ISR. This is called Interrupt LatencyMost computer systems have interrupts generated from the user or the computer system. It would be convenient to handle interrupts from Banker's Algorithm itself. We have taken the approach of handling interrupts with a common queue which can be accessed from the algorithm to check for interrupts and handle it accordingly.

Deadlock Avoidance with Modified Bankers Algorithm to handle interrupts

CSE-2005
Operating Systems
(J Component)
Final Project Report

Project Submitted by:

18BCE0715 - Sanjit C K S

Under the guidance of Dr. Santhi H
Submitted in Partial Fulfilment of the degree of
B.tech in
Computer Science and Engineering, School of Computer Science and Engineering

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Table of Contents

Introduction:

A set of process is in a deadlock state if each process in the set is waiting for an event that can be caused by only another process in the set. In other words, each member of the set of deadlock processes is waiting for a resource that can be released only by a deadlock process. None of the processes can run, none of them can release any resources, and none of them can be awakened. It is important to note that the number of processes and the number and kind of resources possessed and requested are unimportant. The resources may be either physical (Printers, Tape Drivers, etc.) or logical(Files, Semaphores, and Monitors ,etc.) . A process may utilise a resource in the sequence Request, Use and Release. These operations are accomplished by Wait and Signal. A set of processes is in deadlock state when every process in the set of waiting for an event that can be caused only by another process in the set. Deadlock is detected by the wait-for-graph. Deadlock is removed by different mechanisms like deadlock prevention, deadlock handling and deadlock recovery.

Existing Algorithm:

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.’n’ refers to the number of processes in the system and ‘m’, number of resources types.
The “Available” data structure is an array of size ‘m’ indicating the number of available resources of each type.Available[j] = k means there are ‘k’ instances of resource type Rj. The “Max” data structure is a two dimensional array of size ‘n*m’ that defines the maximum demand of each process in a system. Max[i][j] = k means process Pi may request at most ‘k’ instances of resource type Rj.
The “Allocation” data structure is also a two dimensional array of size ‘n*m’ that defines the number of resources of each type currently being allocated to each process.Allocation[i, j] = k means process Pi is currently allocated ‘k’ instances of resource type Rj. The Need Matrix indicates the remaining resource need of each process.Need [i,   j] = k means process Pi currently need ‘k’ instances of resource type Rj for its execution.
Need [i,   j] = Max [i,   j] – Allocation [i,   j]
Banker’s algorithm consists of Safety algorithm and Resource request algorithm

Safety Algorithm:

```
1) Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively.
Initialize: Work = Available
Finish[i] = false; for i=1, 2, 3, 4….n
2) Find an i such that both
a) Finish[i] = false
b) $Need_i$ <= Work
if no such i exists, then the system is in a safe state
3) Work = Work + $Allocation_i$[i]
Finish[i] = false and Needi <= Work
4) if Finish [i] = true for all i
then the system is in a safe state
```

Resource-Request Algorithm

Let $Request_i$ be the request array for process Pi. $Request_i$ [j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken:

1) If $Request_i <= Need_i$

If not, raise an error condition, since the process has exceeded its maximum claim.

2) Check If $Request_i <= Available$

If not, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as
follows:

$Available = Available – Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i– Request_i$

Proposed Algorithm

Modified Bankers Algorithm to Check for Interrupts and its simulation:

Main.c

```
Start
Input Current State of the system processes
Initialise an interrupt wait queue
Create a new thread : Call Modified Bankers Algorithm
Create a new thread : Call Random Interrupt Generation
Wait for modified bankers algorithm to finish executing
Cancel Interrupt Generation thread when modified bankers thread finishes finding safe
sequence
Finish
```

Modified Bankers Algorithm - bakers_process.c

```
1.Start
2.Initialise Current State of System - Allocation, Max Claim, Available
3.Calculate Need Matrix
4.Iterate through processes to find next safe process to execute:
        4.1. Check if interrupts exist by checking if interrupt queue is not empty
        4.2. If no interrupts exist resume safe sequence calculation
        4.3. If interrupts exist : Handle the Interrupts by iterating through the interrupt queue:
                4.3.1. Check if interrupt is not already checked and required resources <
available
                4.3.2. If req_resources <= available print "successfully executed interrupt" and
                        delete it  from interrupt wait queue
                4.3.3. If req_resources > available print "failed to execute and interrupt is pushed
                                back"   and re enqueue the interrupt to the wait queue
                4.3.4. Go back to step 3 to check for new safe sequence process
5. Print Final safe sequence
```

Random Interrupt Generation - interrupt_create.c

1. Start
2. Run an infinite loop
3. Randomly Create an Interrupt with an ID and required resources
4. At random intervals of time Populate the interrupt wait queue by enqueueing the interrupt created.

Other Modules and Files:

Common.c

Printing of all the statuses is done through this file and its function. The logs are maintained as a register and written to a file - “bankerinprocess.csv”. This file is converted to html via a python code - “graph.html”. This file is called and opened in a local browser during the programs runtime via the shell script exec.sh

Queue.c

Contains the interrupt_wait queue and all functionalities of the interrupt_wait queue. Mutex is used with enQueue() function since its used from both interrupt_create thread and bankers_process thread.

Exec.sh

Shell script to:
Compiles the program.
Run the program.
Opens the log file - graph.html , in local browser.

bankerintdataprocessing.py
Converts the file “bankerinprocess.csv” to “graph.html”

Series of Events.txt

```
INTERRUPT_THREAD_START,TH_ID
PROCESS_THREAD_START,TH_ID
FIND_SAFE_SEQUENCE_TO_EXECUTE - AND PRINT IT
CHECKING_INT_QUEUE
SUCCESS:INTERRUPT_DETECTED
    CHECKING_RESOURCE_REQ_OF , INT_ID
    DISPLAY_AVAIL[]
    DISPLAY_RESOURSE_REQ[]
    SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_AND_FREED,INT_ID
    FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAIN,INT_ID
FAILED:NO_INTERRUPT_DETECTED
FINAL_SAFE_SEQUENCE_ORDER
INTERRUPT_THREAD_END,TH_ID
PROCESS_THREAD_END,TH_ID
```

Source Code

bankers_interrupt_main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include "common.h"
#include "queue.h"
#include "interrupt_create.h"
#include "bankers_process.h"

int main()
{
   pthread_t thread_banker, thread_interrupt;

   logfile = fopen(LOG_FILE, "w");
   if (logfile == NULL)
   {
      fprintf(stderr, "error opening logfile");
      return 0;
   }

   fprintf(logfile, "Thread Id, Timestamp in Microsecs, Event, Detail\n");

   memset(buf, 0, 99);
   sprintf(buf, "%d", (int)pthread_self());
   log_message("START_PROCESSING", buf);

   pthread_create(&thread_banker, NULL, process_main, NULL);
   memset(buf, 0, 99);
   sprintf(buf, "%d", (int)thread_banker);
   log_message("SAFE_PROCESS_THREAD_START", buf);

   pthread_create(&thread_interrupt, NULL, interrupt_main, NULL);
   memset(buf, 0, 99);
   sprintf(buf, "%d", (int)thread_interrupt);
   log_message("INTERRUPT_THREAD_START", buf);

   pthread_join(thread_banker, NULL);
   pthread_cancel(thread_interrupt);
   log_message("END_PROCESSING", "");

   fclose(logfile);
   printf("AFTER_THREADING\n");
   exit(0);
}
```

queue.h

```
#ifndef QUEUE_H_INCLUDED
#define QUEUE_H_INCLUDED
#define SIZE 10

extern int front;
extern int rear;
extern int interrupt_wait[SIZE][4];
// void init();
void enQueue(int interrupt_resources[]);
int* deQueue();
int topInterruptId();
int getSize();
int isEmpty();
int isFull();
void display();
#endif
```

queue.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "queue.h"
#include "common.h"

//int interrupt_resources[SIZE][3];//3 is the number of resources
int front = -1;
int rear = -1;
int interrupt_wait[SIZE][4];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void display()
{
    int i, j;
    printf("\n[%d] DISPLAYING INTERRUPT\n", (int)pthread_self());
    if (isEmpty())
        printf(" \n[%d] QUEUE IS EMPTY!\n", (int)pthread_self());
    else
    {
        printf("\n[%d] FRONT : %d\n", (int)pthread_self(), front);
        printf("[%d] INTERRUPT_WAIT: ", (int)pthread_self());
        for (i = front; i != rear; i = (i + 1) % SIZE)
        {
            for (j = 0; j < 4; j++)
            {
                printf("%d \t", interrupt_wait[i][j]);
            }
            printf("\n");
        }
        for (j=0;j<4;j++)
        {
            printf("%d\t",interrupt_wait[rear][j]);
        }
        printf("\n");
        printf("\n[%d] REAR :%d\n", (int)pthread_self(), rear);
        fflush(stdout);
    }
}

int topInterruptId()
{
    if (isEmpty())
    {
        return (-1);
    }
    else
    {
        return interrupt_wait[front][0];
    }
}

int getSize()
{
    if (front == -1 && rear == -1)
    {
        return 0;
    }
    else if (front == rear)
    {
        return 1;
    }
    else
    {
        return abs(front - rear);
    }
}
int isFull()
{
    if ((front == rear + 1) || (front == 0 && rear == SIZE - 1))
        return 1;
    return 0;
}
int isEmpty()
{
    if (front == -1)
        return 1;
    return 0;
}
void enQueue(int interrupt_resources[])
{
    int i = 0;

    if (isFull())
```

```
        printf("\n[th q] INTERRUPT QUEUE IS FULL! \n");
    else
    {
        pthread_mutex_lock(&mutex);
        if (front == -1)
            front = 0;
        rear = (rear + 1) % SIZE;
        for (i = 0; i < 4; i++)
        {

            interrupt_wait[rear][i] = interrupt_resources[i];
        }
        pthread_mutex_unlock(&mutex);
        display();
        printf("\n[th q] INTERRUPT INSERTED IS : %d", interrupt_resources[0]);
    }
}
int *deQueue()
{
    int i, send_top_arr[4];
    if (isEmpty())
    {
        printf("\n[th q] INTERRUPT QUEUE IS EMPTY! \n");
        return NULL;
    }
    else
    {
        for (i = 0; i < 4; i++)
        {
            send_top_arr[i] = interrupt_wait[front][i];
        }
        printf("\n[th q] INTERRUPT DEQUEUED IS : %d \n", interrupt_wait[front][0]);
        if (front == rear)
        {
            front = -1;
            rear = -1;
        } /* Q has only one element, so we reset the queue after dequeing it. ? */
        else
        {
            front = (front + 1) % SIZE;
        }
        return send_top_arr;
    }
}
```

bankers_process.h

```
#ifndef BANKERS_PROCESS_H_INCLUDED
#define BANKERS_PROCESS_H_INCLUDED
void *process_main(void *arg);
#endif
```

bankers_process.c

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "queue.h"
#include "common.h"

void *process_main(void *arg)
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k, x, interrupt_choice, processed_first, temp, interrupt_count = 0, not_en = 0;
   int *temp_ptr;
   int temp_arr[4];
   char tbuf[100];
   n = 5;                        // Number of processes
   m = 3;                        // Number of resources
   int alloc[5][3] = {{0, 1, 0},  // P0    // Allocation Matrix
                      {2, 0, 0},  // P1
                      {3, 0, 2},  // P2
                      {2, 1, 1},  // P3
                      {0, 0, 2}}; // P4

   int max[5][3] = {{7, 5, 3},  // P0    // MAX Matrix
                    {3, 2, 2},  // P1
                    {9, 0, 2},  // P2
                    {2, 2, 2},  // P3
                    {4, 3, 3}}; // P4

   int avail[3] = {3, 3, 2}; // Available Resources

   int f[n], ans[n], ind = 0;
   for (k = 0; k < n; k++)
   {
      f[k] = 0;
   }
   int need[n][m];
   for (i = 0; i < n; i++)
   { //creation of need matrix
      for (j = 0; j < m; j++)
         need[i][j] = max[i][j] - alloc[i][j];
   }
   int y = 0;
   for (k = 0; k < 5; k++)
   { //
the loop below doesnt guarentee that all the processes are executed, so the worst case senari
o for a safe state is that only one process is executed in one iteration
      for (i = 0; i < n; i++)
      { //for each process
```

```
        if (f[i] == 0)
        { //thats has not been executed yet

            int flag = 0;
            for (j = 0; j < m; j++)
            { //check for all resources if need is not greater than available
                if (need[i][j] > avail[j])
                { //if it is greater then it cant be executed so set flag to 1
                    flag = 1;
                    break;
                }
            }

            if (flag == 0)
            {                //if the particular process can be executed (need<avail) - execute it
                ans[ind++] = i; //
seperate index ind is used and incremented after assign/"execute" process
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y]; //resource is freed up
                f[i] = 1;                 //set finish to 1
                sleep(1);
                memset(buf, 0, 99);
                sprintf(buf, "P%d", i);
                log_message("PROCESS_EXECUTED", buf);
                //now check if the interrupt wait queue is not empty
                log_message("CHECKING_INT_QUEUE", "");
                if (isEmpty())
                {
                    log_message("FAILED:NO_INTERRUPT_DETECTED", "");
                }
                else
                {
                    log_message("SUCCESS:INTERRUPT_DETECTED", "");
                    //check if the interrupt can be granted
                    processed_first = -1;
                    while (!isEmpty())
                    {
                        if (topInterruptId() == processed_first)
                        {
                            log_message("PROCESSED_ALL_POSSIBLE_WAITING_INTERRUPTS", " "
);

                            break;
                        }

                        temp_ptr = deQueue();
                        for (x = 0; x < 4; x++)
                        {
                            temp_arr[x] = *(int *)(temp_ptr + x);
                        }

                        memset(buf, 0, 99);
                        sprintf(buf, "[");
                        for (x = 0; x < 3; x++)
                        {
                            memset(tbuf, 0, 99);
                            sprintf(tbuf, "%d ", avail[x]);
                            strcat(buf, tbuf);
                        }
                        strcat(buf, "]");
                        log_message("CHECKING_AVLBL_RESOURCE", buf);

                        memset(buf, 0, 99);
                        sprintf(buf, "[");
                        for (x = 0; x < m + 1; x++)
                        {
                            memset(tbuf, 0, 99);
                            sprintf(tbuf, "%d ", temp_arr[x]);
                            strcat(buf, tbuf);
                        }
                        strcat(buf, "]");
                        log_message("CHECKING_INT_NEED", buf);

                        //
couldnt find a way to convert arrays of avail and temp_arr into strings and call log_message
                        not_en = 0;
                        for (x = 1; x < m + 1; x++)
                        {
                            if (temp_arr[x] > avail[x - 1])
                            {
                                not_en = 1;
                                break;
                            }
                        }
                        if (not_en == 1)
                        {
                            memset(buf, 0, 99);
                            sprintf(buf, "%d", temp_arr[0]);
                            log_message("FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAI
N", buf);
                            if (processed_first == -1)
                                processed_first = temp_arr[0];
                            enQueue(temp_arr);
                        }

                        if (not_en == 0)
                        {
                            memset(buf, 0, 99);
                            sprintf(buf, "%d", temp_arr[0]);
                            log_message("SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_U
SED_AND_FREED", buf);
                        }
                    }
                }
            }
        }
    }
}
/*if (!isEmpty()){
printf("\nInterrupt %d could not be granted because required resources are too high!!
\n",interrupt_wait[front][0]);
} */
printf("\nSAFE SEQUENCE GENERATED FOR THE PROCESSES IN THE CURRENT STATE I
S : \n");
memset(buf, 0, 99);
for (i = 0; i < n - 1; i++)
{
    memset(tbuf, 0, 99);
    sprintf(tbuf, " P%d ->", ans[i]);
    strcat(buf, tbuf);
}
memset(tbuf, 0, 99);
sprintf(tbuf, " P%d", ans[n - 1]);
strcat(buf, tbuf);
log_message("SAFE_SEQUENCE_COMPUTED", buf);

return 0;
}
```

interrupt_create.h

```
#ifndef INTERRUPT_MAIN_H_INCLUDED
#define INTERRUPT_MAIN_H_INCLUDED
void *interrupt_main(void *arg);
#endif
```

interrupt_create.c

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include "queue.h"
#include <time.h>
#include "common.h"

void *interrupt_main(void *arg)
{
    struct timeval rt;
    int interrupt_resources[4];
    int i, temp_rand, int_id_start = 100;
    char lbuf[100];
    char tbuf[10];
    while (1)
    {
        memset(lbuf, 0, 99);
        temp_rand = 0;
        gettimeofday(&rt, NULL);
        srand(ct.tv_sec * 1000000 + ct.tv_usec);
        interrupt_resources[0] = int_id_start;

        sprintf(lbuf, "[ %d ", int_id_start);
        for (i = 1; i < 4; i++)
        {
            memset(tbuf, 0, 9);
            interrupt_resources[i] = rand() % 7;
            sprintf(tbuf, "%d ", interrupt_resources[i]);
            strcat(lbuf, tbuf);
        }
        strcat(lbuf, "]");
        log_message("INTERRUPT_QUEUED", lbuf);
        fflush(stdout);
        enQueue(interrupt_resources);
        while (temp_rand == 0)
        {
            temp_rand = rand() % 3;
        }
        sleep(temp_rand);
        int_id_start++;
    }
    return 0;
}
```

common.h

```
#ifndef COMMON_H_INCLUDED
#define COMMON_H_INCLUDED
#include <sys/time.h>
#include <string.h>

#define LOG_FILE "bankerintprocess.csv"

struct timeval ct;
void log_message(char event[], char detail[]);
char buf[100];
FILE *logfile;
#endif
```

common.c

```
#include <stdio.h>
#include "common.h"
#include <pthread.h>

void log_message(char event[], char detail[])
{
    gettimeofday(&ct, NULL);
    printf("\nLOG: [%d] %ld %s-
%s\n", (int)pthread_self(), ct.tv_sec * 1000000 + ct.tv_usec, event, detail);
    fprintf(logfile, "%d,%ld,%s,
%s\n", (int)pthread_self(), ct.tv_sec * 1000000 + ct.tv_usec, event, detail);
}
```

bankerintdataprocessing.py

```
#! /usr/bin/env python3
import csv
import pandas
import numpy as np

input_file = "./bankerintprocess.csv"

df = pandas.read_csv(input_file)

with open('graph.html', 'w') as fo:
    df.to_html(fo)
```

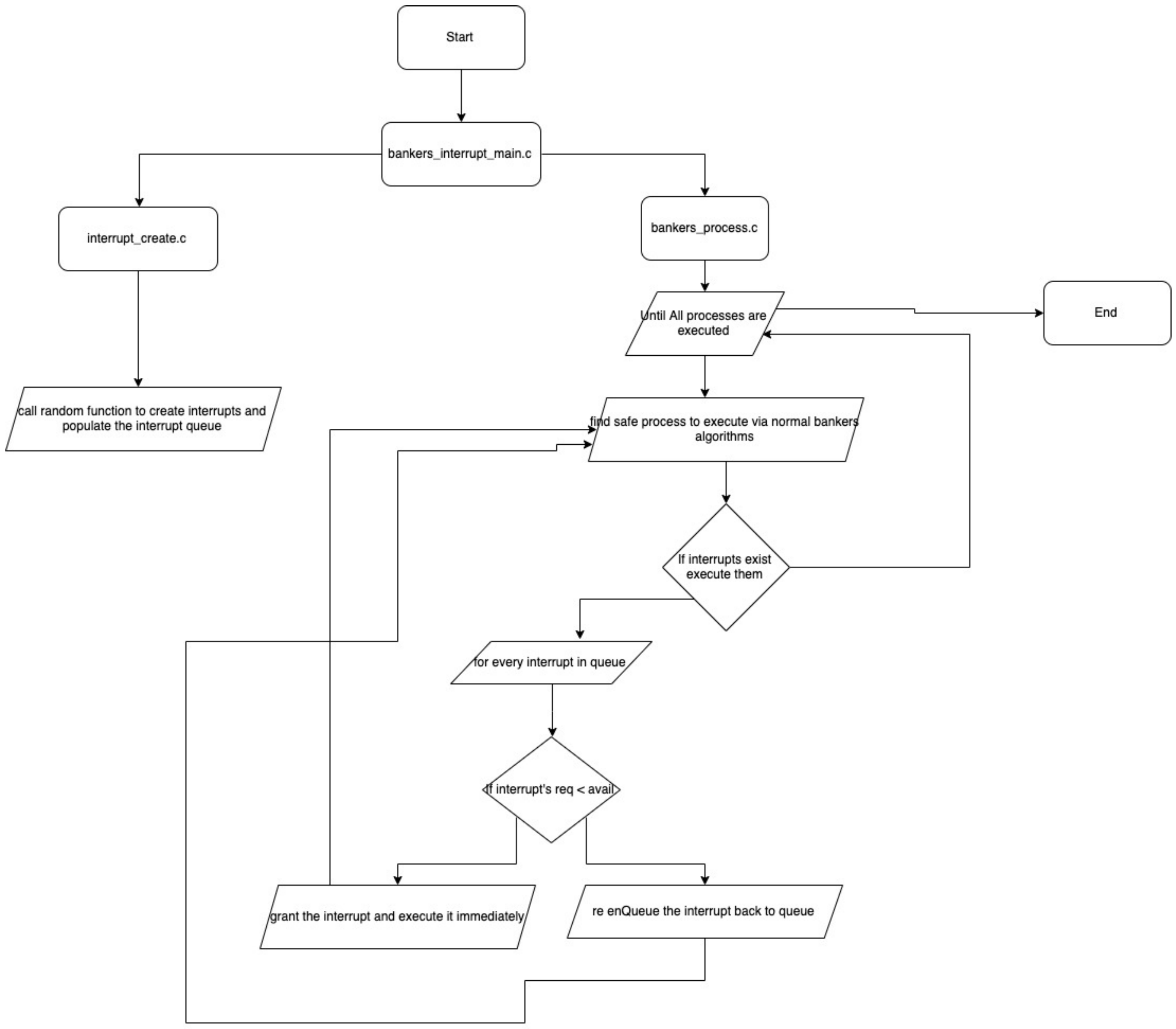
app.js

```
const http = require('http'),
    fs = require('fs');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Welcome To Banker Interrupt Project Report. Go to /graph.html to view the report');
});

server.listen(port, hostname, () => {
  console.log();
});

fs.readFile('./graph.html', function (err, html) {
    if (err) {
        throw err;
    }
    http.createServer(function(request, response) {
        response.writeHeader(200, {"Content-Type": "text/html"});
        response.write(html);
        response.end();
    }).listen(3000);
});
```

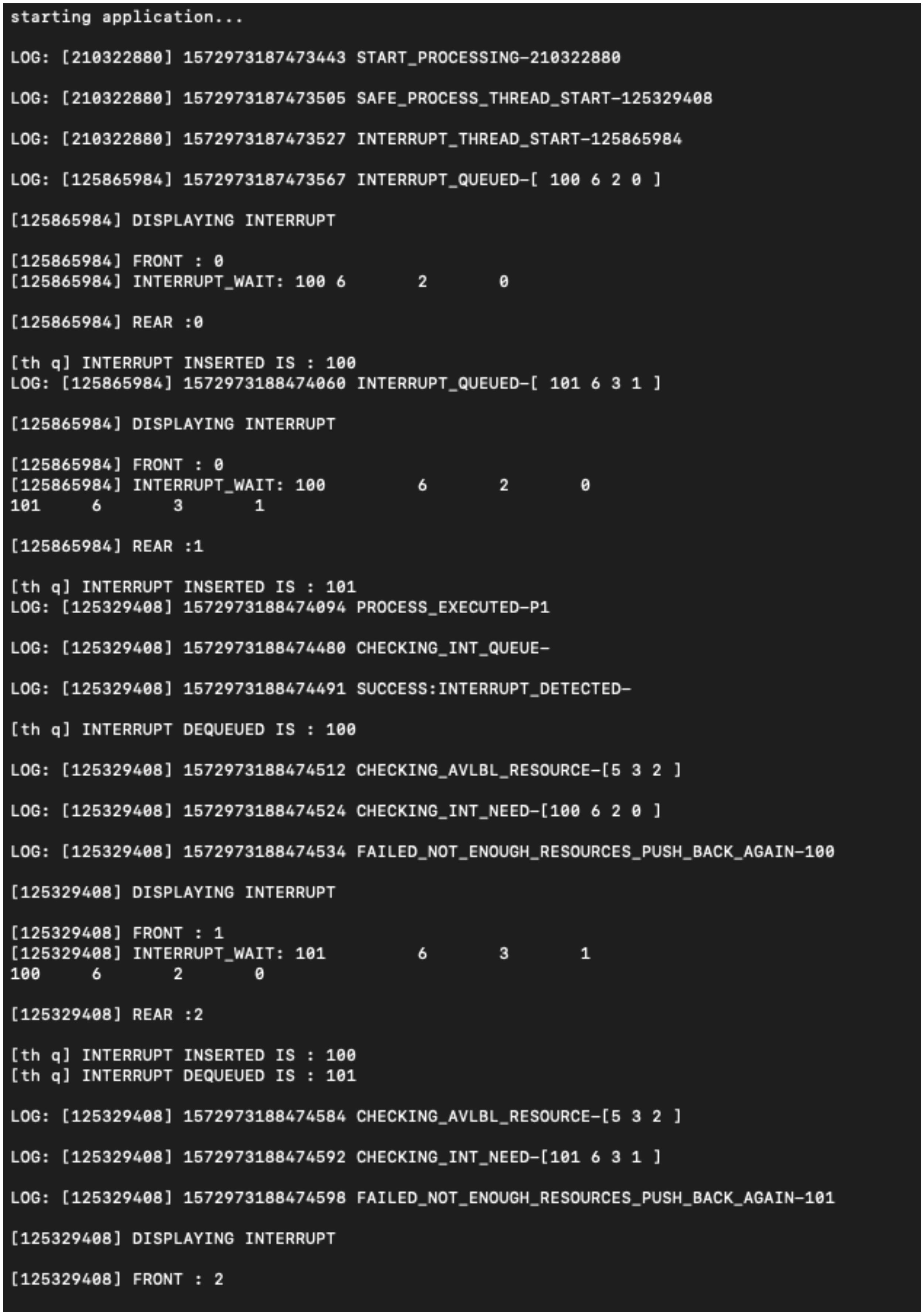
exec.sh

```
echo "start compiling c program"
echo
gcc  bankers_interrupt_main.c bankers_process.c interrupt_create.c  queue.c common.c

echo "starting application..."
./a.out
echo "application finished=========================="

python bankerintdataprocessing.py

echo "kill if server running at port 3000 if any"
ps aux | grep 3000 | grep -v grep | awk '{print $2}' | uniq | xargs kill -9

if which node > /dev/null
then
   echo "node is installed, skipping installation..."
else
   echo "Installing node..."
   npm install node
fi

echo
echo "starting node server"
node app.js 3000 &

echo "started node server in detached mode"
echo
open http://localhost:3000
echo
echo "opened report in your favorite browser”
```

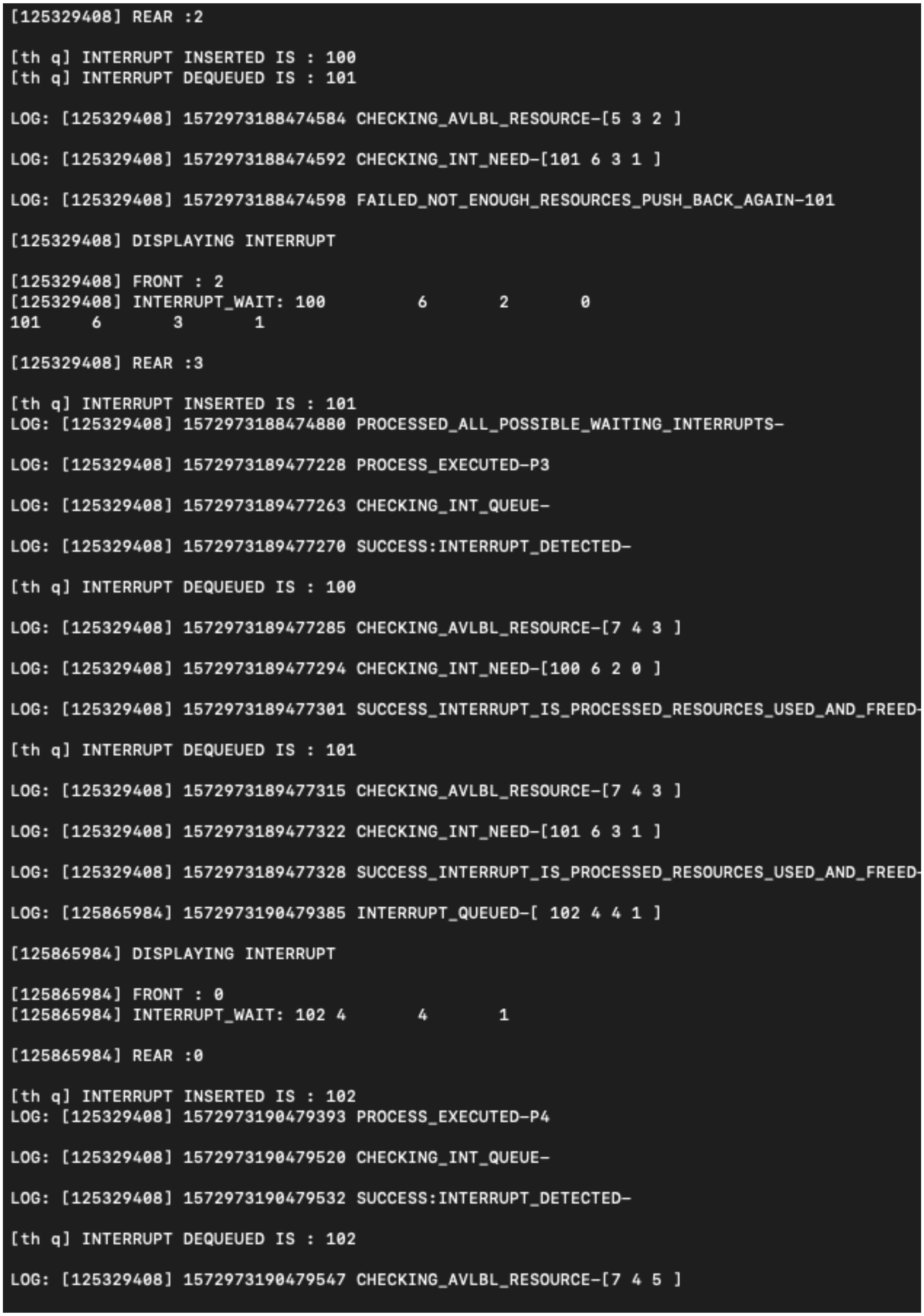
Flow Chart of Proposed Model


Start
bankers_interrupt_main.c
interrupt_create.c
bankers_process.c
Until All processes are executed
End
call random function to create interrupts and populate the interrupt queue
find safe process to execute via normal bankers algorithms
If interrupts exist execute them
for every interrupt in queue
If interrupt's req < avail
grant the interrupt and execute it immediately
re enQueue the interrupt back to queue

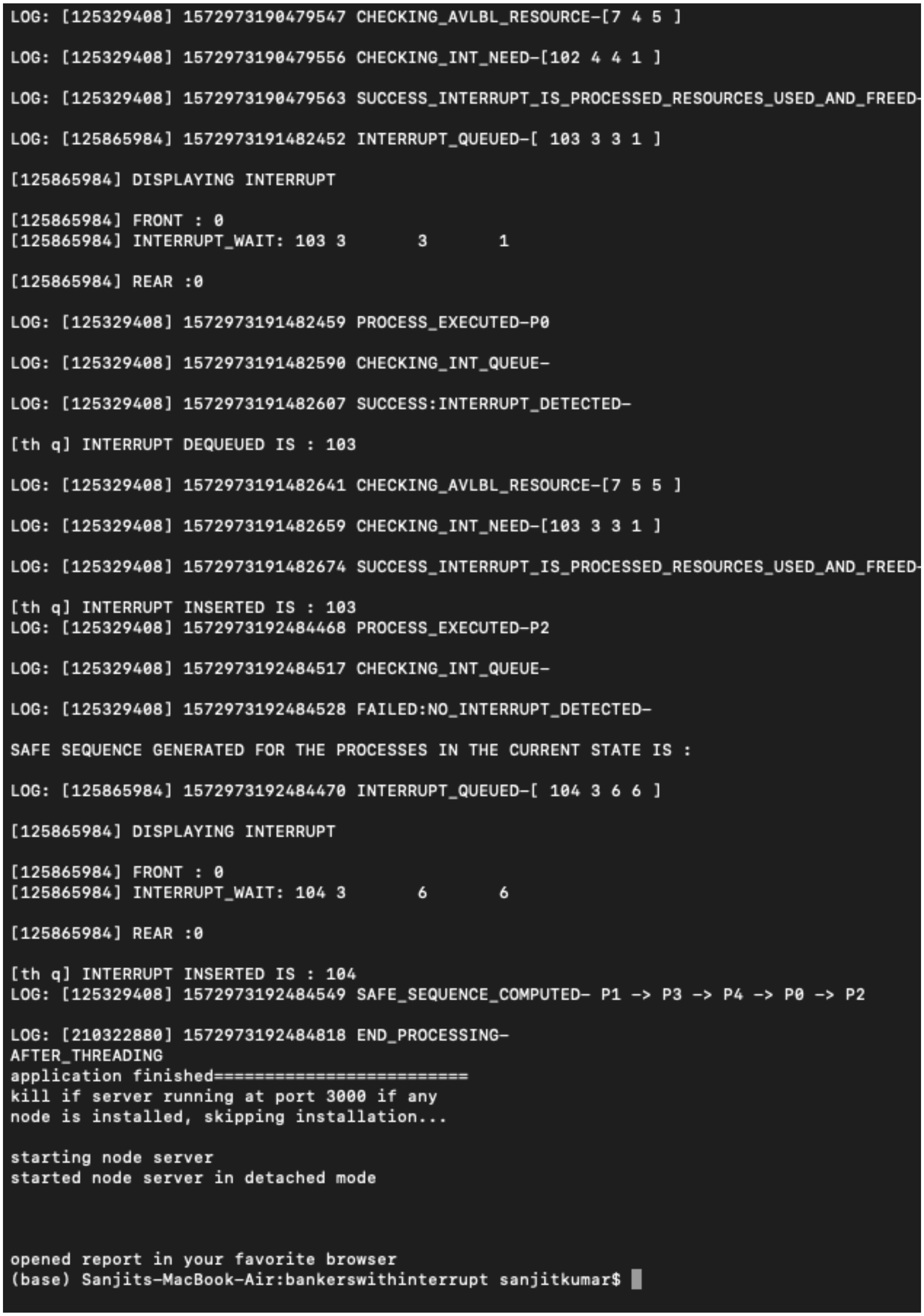
Results



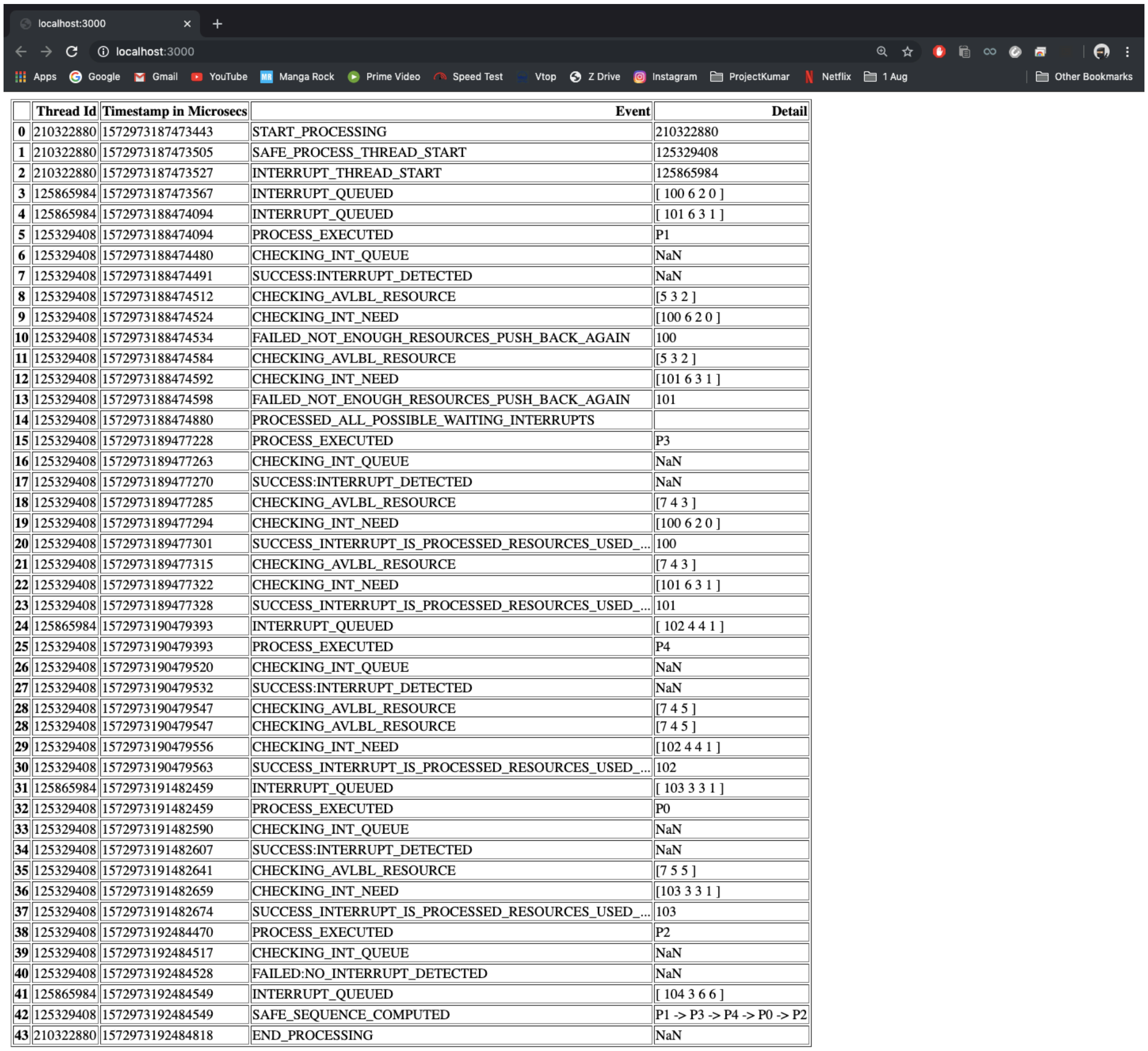
```
starting application...

LOG: [210322880] 1572973187473443 START_PROCESSING-210322880

LOG: [210322880] 1572973187473505 SAFE_PROCESS_THREAD_START-125329408

LOG: [210322880] 1572973187473527 INTERRUPT_THREAD_START-125865984

LOG: [125865984] 1572973187473567 INTERRUPT_QUEUED-[ 100 6 2 0 ]

[125865984] DISPLAYING INTERRUPT

[125865984] FRONT : 0
[125865984] INTERRUPT_WAIT: 100 6       2       0

[125865984] REAR :0

[th q] INTERRUPT INSERTED IS : 100
LOG: [125865984] 1572973188474060 INTERRUPT_QUEUED-[ 101 6 3 1 ]

[125865984] DISPLAYING INTERRUPT

[125865984] FRONT : 0
[125865984] INTERRUPT_WAIT: 100         6       2       0
101     6       3       1

[125865984] REAR :1

[th q] INTERRUPT INSERTED IS : 101
LOG: [125329408] 1572973188474094 PROCESS_EXECUTED-P1

LOG: [125329408] 1572973188474480 CHECKING_INT_QUEUE-

LOG: [125329408] 1572973188474491 SUCCESS:INTERRUPT_DETECTED-

[th q] INTERRUPT DEQUEUED IS : 100

LOG: [125329408] 1572973188474512 CHECKING_AVLBL_RESOURCE-[5 3 2 ]

LOG: [125329408] 1572973188474524 CHECKING_INT_NEED-[100 6 2 0 ]

LOG: [125329408] 1572973188474534 FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAIN-100

[125329408] DISPLAYING INTERRUPT

[125329408] FRONT : 1
[125329408] INTERRUPT_WAIT: 101         6       3       1
100     6       2       0

[125329408] REAR :2

[th q] INTERRUPT INSERTED IS : 100
[th q] INTERRUPT DEQUEUED IS : 101

LOG: [125329408] 1572973188474584 CHECKING_AVLBL_RESOURCE-[5 3 2 ]

LOG: [125329408] 1572973188474592 CHECKING_INT_NEED-[101 6 3 1 ]

LOG: [125329408] 1572973188474598 FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAIN-101

[125329408] DISPLAYING INTERRUPT

[125329408] FRONT : 2
```



```
[125329408] REAR :2

[th q] INTERRUPT INSERTED IS : 100
[th q] INTERRUPT DEQUEUED IS : 101

LOG: [125329408] 1572973188474584 CHECKING_AVLBL_RESOURCE-[5 3 2 ]

LOG: [125329408] 1572973188474592 CHECKING_INT_NEED-[101 6 3 1 ]

LOG: [125329408] 1572973188474598 FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAIN-101

[125329408] DISPLAYING INTERRUPT

[125329408] FRONT : 2
[125329408] INTERRUPT_WAIT: 100         6       2       0
101     6       3       1

[125329408] REAR :3

[th q] INTERRUPT INSERTED IS : 101
LOG: [125329408] 1572973188474880 PROCESSED_ALL_POSSIBLE_WAITING_INTERRUPTS-

LOG: [125329408] 1572973189477228 PROCESS_EXECUTED-P3

LOG: [125329408] 1572973189477263 CHECKING_INT_QUEUE-

LOG: [125329408] 1572973189477270 SUCCESS:INTERRUPT_DETECTED-

[th q] INTERRUPT DEQUEUED IS : 100

LOG: [125329408] 1572973189477285 CHECKING_AVLBL_RESOURCE-[7 4 3 ]

LOG: [125329408] 1572973189477294 CHECKING_INT_NEED-[100 6 2 0 ]

LOG: [125329408] 1572973189477301 SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_AND_FREED

[th q] INTERRUPT DEQUEUED IS : 101

LOG: [125329408] 1572973189477315 CHECKING_AVLBL_RESOURCE-[7 4 3 ]

LOG: [125329408] 1572973189477322 CHECKING_INT_NEED-[101 6 3 1 ]

LOG: [125329408] 1572973189477328 SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_AND_FREED

LOG: [125865984] 1572973190479385 INTERRUPT_QUEUED-[ 102 4 4 1 ]

[125865984] DISPLAYING INTERRUPT

[125865984] FRONT : 0
[125865984] INTERRUPT_WAIT: 102 4       4       1

[125865984] REAR :0

[th q] INTERRUPT INSERTED IS : 102
LOG: [125329408] 1572973190479393 PROCESS_EXECUTED-P4

LOG: [125329408] 1572973190479520 CHECKING_INT_QUEUE-

LOG: [125329408] 1572973190479532 SUCCESS:INTERRUPT_DETECTED-

[th q] INTERRUPT DEQUEUED IS : 102

LOG: [125329408] 1572973190479547 CHECKING_AVLBL_RESOURCE-[7 4 5 ]
```



```
LOG: [125329408] 1572973190479547 CHECKING_AVLBL_RESOURCE-[7 4 5 ]

LOG: [125329408] 1572973190479556 CHECKING_INT_NEED-[102 4 4 1 ]

LOG: [125329408] 1572973190479563 SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_AND_FREED-

LOG: [125865984] 1572973191482452 INTERRUPT_QUEUED-[ 103 3 3 1 ]

[125865984] DISPLAYING INTERRUPT

[125865984] FRONT : 0
[125865984] INTERRUPT_WAIT: 103 3       3       1

[125865984] REAR :0

LOG: [125329408] 1572973191482459 PROCESS_EXECUTED-P0

LOG: [125329408] 1572973191482590 CHECKING_INT_QUEUE-

LOG: [125329408] 1572973191482607 SUCCESS:INTERRUPT_DETECTED-

[th q] INTERRUPT DEQUEUED IS : 103

LOG: [125329408] 1572973191482641 CHECKING_AVLBL_RESOURCE-[7 5 5 ]

LOG: [125329408] 1572973191482659 CHECKING_INT_NEED-[103 3 3 1 ]

LOG: [125329408] 1572973191482674 SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_AND_FREED-

[th q] INTERRUPT INSERTED IS : 103
LOG: [125329408] 1572973192484468 PROCESS_EXECUTED-P2

LOG: [125329408] 1572973192484517 CHECKING_INT_QUEUE-

LOG: [125329408] 1572973192484528 FAILED:NO_INTERRUPT_DETECTED-

SAFE SEQUENCE GENERATED FOR THE PROCESSES IN THE CURRENT STATE IS :

LOG: [125865984] 1572973192484470 INTERRUPT_QUEUED-[ 104 3 6 6 ]

[125865984] DISPLAYING INTERRUPT

[125865984] FRONT : 0
[125865984] INTERRUPT_WAIT: 104 3       6       6

[125865984] REAR :0

[th q] INTERRUPT INSERTED IS : 104
LOG: [125329408] 1572973192484549 SAFE_SEQUENCE_COMPUTED- P1 -> P3 -> P4 -> P0 -> P2

LOG: [210322880] 1572973192484818 END_PROCESSING-
AFTER_THREADING
application finished========================
kill if server running at port 3000 if any
node is installed, skipping installation...

starting node server
started node server in detached mode



opened report in your favorite browser
(base) Sanjits-MacBook-Air:bankerswithinterrupt sanjitkumar$
```



	Thread Id	Timestamp in Microsecs	Event	Detail
0	210322880	1572973187473443	START_PROCESSING	210322880
1	210322880	1572973187473505	SAFE_PROCESS_THREAD_START	125329408
2	210322880	1572973187473527	INTERRUPT_THREAD_START	125865984
3	125865984	1572973187473567	INTERRUPT_QUEUED	[100 6 2 0]
4	125865984	1572973188474094	INTERRUPT_QUEUED	[101 6 3 1]
5	125329408	1572973188474094	PROCESS_EXECUTED	P1
6	125329408	1572973188474480	CHECKING_INT_QUEUE	NaN
7	125329408	1572973188474491	SUCCESS:INTERRUPT_DETECTED	NaN
8	125329408	1572973188474512	CHECKING_AVLBL_RESOURCE	[5 3 2]
9	125329408	1572973188474524	CHECKING_INT_NEED	[100 6 2 0]
10	125329408	1572973188474534	FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAIN	100
11	125329408	1572973188474584	CHECKING_AVLBL_RESOURCE	[5 3 2]
12	125329408	1572973188474592	CHECKING_INT_NEED	[101 6 3 1]
13	125329408	1572973188474598	FAILED_NOT_ENOUGH_RESOURCES_PUSH_BACK_AGAIN	101
14	125329408	1572973188474880	PROCESSED_ALL_POSSIBLE_WAITING_INTERRUPTS	
15	125329408	1572973189477228	PROCESS_EXECUTED	P3
16	125329408	1572973189477263	CHECKING_INT_QUEUE	NaN
17	125329408	1572973189477270	SUCCESS:INTERRUPT_DETECTED	NaN
18	125329408	1572973189477285	CHECKING_AVLBL_RESOURCE	[7 4 3]
19	125329408	1572973189477294	CHECKING_INT_NEED	[100 6 2 0]
20	125329408	1572973189477301	SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_...	100
21	125329408	1572973189477315	CHECKING_AVLBL_RESOURCE	[7 4 3]
22	125329408	1572973189477322	CHECKING_INT_NEED	[101 6 3 1]
23	125329408	1572973189477328	SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_...	101
24	125865984	1572973190479393	INTERRUPT_QUEUED	[102 4 4 1]
25	125329408	1572973190479393	PROCESS_EXECUTED	P4
26	125329408	1572973190479520	CHECKING_INT_QUEUE	NaN
27	125329408	1572973190479532	SUCCESS:INTERRUPT_DETECTED	NaN
28	125329408	1572973190479547	CHECKING_AVLBL_RESOURCE	[7 4 5]
28	125329408	1572973190479547	CHECKING_AVLBL_RESOURCE	[7 4 5]
29	125329408	1572973190479556	CHECKING_INT_NEED	[102 4 4 1]
30	125329408	1572973190479563	SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_...	102
31	125865984	1572973191482459	INTERRUPT_QUEUED	[103 3 3 1]
32	125329408	1572973191482459	PROCESS_EXECUTED	P0
33	125329408	1572973191482590	CHECKING_INT_QUEUE	NaN
34	125329408	1572973191482607	SUCCESS:INTERRUPT_DETECTED	NaN
35	125329408	1572973191482641	CHECKING_AVLBL_RESOURCE	[7 5 5]
36	125329408	1572973191482659	CHECKING_INT_NEED	[103 3 3 1]
37	125329408	1572973191482674	SUCCESS_INTERRUPT_IS_PROCESSED_RESOURCES_USED_...	103
38	125329408	1572973192484470	PROCESS_EXECUTED	P2
39	125329408	1572973192484517	CHECKING_INT_QUEUE	NaN
40	125329408	1572973192484528	FAILED:NO_INTERRUPT_DETECTED	NaN
41	125865984	1572973192484549	INTERRUPT_QUEUED	[104 3 6 6]
42	125329408	1572973192484549	SAFE_SEQUENCE_COMPUTED	P1 -> P3 -> P4 -> P0 -> P2
43	210322880	1572973192484818	END_PROCESSING	NaN

Reference to Research Papers

[1] Ahmed Nazeem and Spyros Reveliotis, “Designing Compact and Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems Through Classification Theory- The Nonlinear Case”, *Ieee transactions on automatic control,* volume 57, Issue7, pages 1670-1684

[2] Passent M El-Kafrawy, “Graphical Deadlock Avoidance”, *International Conference on Advances in Computing, Control, and Telecommunication Technologies 2009*, volume 1,issue 1, pp. 308-312

[3]Luis Mejía-Ricart and Aspen Olmsted ,”Avoiding unnecessary deaths, Drag-back, a deadlock avoidance model”,*12th International Conference for Internet Technology and Secured Transactions,* pp. 472-474

[4] Gonzalo Zarza, Diego Lugones, Daniel Franco, and Emilio Luque , “Deadlock Avoidance for Interconnection Networks with Multiple Dynamic Faults”, *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing,* pp. 276-280

[5] Yu Zhen* and Su Xiaohong, Ma Peijun ,“Scrider: Using Single Critical Sections to Avoid Deadlocks”, *2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control,* pp. 1000-1005

Comparison

Name of the Paper	Year	Problem Addressed	Proposed Technique/ Algorithm	Limitations
Designing Compact and Maximally Permissive Deadlock Avoidance Policies for Complex Resource Allocation Systems Through Classification Theory: The Nonlinear Case	2012	Optimising the maximally permissive resource allocation scheme for complex RAS	Classification Theory: The Non-linear case	More Data on Processes Required
Graphical Deadlock Avoidance	2009	Automated Manufacturing Cell (ACM), concurrency con-trol (CC) and communication, deadlock	Graphical Solution - Inter Diagrammatic Reasoning - Using Petri nets and FSM	Graphs have to be calculated and recomputed every step of the way.
Avoiding unnecessary deaths Drag-Back, a deadlock avoidance model	2017	Complete Rollback of transactions in case of wait-die/ wound-wait irrespective of amount of overlap between the transactions - wasted IO op. and disk writes	Partial rollback of transactions implemented by compensation logs (a WAL)	Not applicable for all kinds of transactions - requires some level of flexibility
Deadlock Avoidance for Interconnection Networks with Multiple Dynamic Faults	2010	Deadlock in Networks.- Due to the long execution times of computationally intensive applications, some interconnection networks may show a MTBF smaller than the execution time of such applications.	The solution to this problem involves developing a deadlock avoidance technique capable of allowing an un-bounded number of direction changes in the routing function.	Routing algorithm becomes more complex and takes more time.

Name of the Paper	Year	Problem Addressed	Proposed Technique/ Algorithm	Limitations
Scrider- Using Single Critical Sections to Avoid Deadlocks	2014	Deadlocks in threading	Hijack Algorithm - controls mutex's lock/unlock functions	Invasive way of doing it - possible program slow down Cannot avoid deadlocks by conditional variables and semaphores

Comparison with Research Papers

The research papers referred above have dealt with the time complexity of DAP algorithms and have accordingly provided solutions to make the DAP algorithm better and more efficient. They have dealt with new theories and explored new areas of database oriented transactions with deadlock avoidance. Policies like the scrider have proposed an invasive algorithm that will hijack the existing systems mutex functions to deal with deadlock. The algorithm we have proposed only adds to the functionality of the regular Dijkstra's bankers algorithm. We have successfully simulated such a situation using multithreaded code with pthreads and its functions. Through bankersWithInterrupt, we have proposed a model for a bankers algorithm to detect interrupts when they are generated and have proposed a solution to handle them accordingly.

Classification Theory:

The paper talks about optimising a maximally permissive deadlock avoidance policy by using the classification theory's non linear case. The authors have extrapolated on an earlier research where they talked about the linear case of the classification theory. This scheme although optimises the DAP and makes it undoubtably better, doesn't add to the normal functionality of bankers algorithm or any DAP. BankersWithInterrupt deals with interrupt handling from the DAP itself.

Graphical Deadlock:

This paper talks about a deadlock avoidance policy that functions based on graphs. It uses petri nets and finite state machines to depict the deadlock situation and uses inter diagrammatic reasoning to deal with it. Again the solution doesn't add to the functionality of the bankers algorithm.

Avoiding deaths via roll backs:

This paper explores the database concepts of transaction and processes in os. It suggests how sacrificing the atomicity of the processes in certain occasions and specific cases where it is not required might lead to increase in performance of the system and avoid deadlocks. This algorithm also doesn’t deal with the case of interrupts in the system.

Interconnection networks and multiple dynamic faults:

In case of networks operating systems due to the long and intensive applications, the mean time between failure can be lesser than execution time. This leads to failures surely and are dealt with by allowing unbounded changes in the routing algorithm. The paper improves performance in network systems but doesn’t deal with interrupts in the DAP.

Scrider:

This paper deals with interrupts and avoids deadlock via a hijack algorithm that takes over the control of the mutex functions for the processes. This is not recommended as it is an invasive way of dealing with the problem. Also again, it doesn’t deal with any incoming interrupts during this.

Conclusion and Future Scope

Through this project we have successfully proposed an algorithm that is a modified and improved Dijkstra's bankers algorithm that can check for interrupts during deadlock safe sequence generation and handle them appropriately. We have also successfully simulated this using pthreads via a multithreaded program with a suitable example scenario. We have used suitable header files, randomise functions and unixstd functions to simulate the example. We have also gone ahead and logged all the statuses to a file which is then converted to a html file - 'graph' by parsing via a python script. This is then displayed on the local browser through a node java script code.

The handling of interrupts in a regular computer happens outside the deadlock avoidance policy. We have suggested a new technique of checking for interrupts through a modified bankers algorithms. This program can be extended to other areas of deadlock avoidance and can be made for user input set of process in the future. It can also be made with dynamic resource allocation. So there is scope for improvement and development in the future. The idea of interrupt handling within bankers has given us the opportunity to try and handle these interrupts in the manner we have done here. So there is much potential for development in different future directions.