ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

“Разработка операционной системы реального времени”
по дисциплине «Архитектура программных систем»

Студент
3530202/90202 Потапова А. М.

Преподаватель Коликова Т. В.

Санкт-Петербург
2021 г.

ЗАДАНИЕ
НА ВЫПОЛНЕНИЕ КУРСОВОГО ПРОЕКТА (КУРСОВОЙ РАБОТЫ)

студенту

группы *3530202/90202* (номер группы) *Потаповой Алине Михайловне* (фамилия, имя, отчество)

1. Тема проекта (работы) *Разработка автоматизированной системы «Модель операционной системы реального времени».*

2. Срок сдачи студентом законченного проекта (работы) *25.05.2022*

3. Исходные данные к проекту (работе)

Тип планировщика: плоский; Алгоритм планировщика: РТ; Управление ресурсами: простые семафоры; Управление событиями: нет; Обработка прерываний: есть, допустимо объявление локальных переменных в обработчике прерываний; Максимальное количество задач: 32; Максимальное количество ресурсов: 16.

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов: введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения).

Введение. Основная часть. Заключение. Приложения

Примерный объем пояснительной записки *10-15* страниц машинописного текста

5. Перечень графического материала (с указанием обязательных чертежей и плакатов) *не предоставляется*

6. Консультанты

7. Дата получения задания: «1» февраля 2022 г.

Руководитель _____ (подпись) *Коликова Т.В.* (инициалы, фамилия)

Задание принял к исполнению _____ (подпись) *Потапова А.М.* (инициалы, фамилия)

26.04.2022 (дата)

Оглавление

Постановка задачи

Необходимо реализовать модель операционной системы реального времени, обладающей следующими свойствами:

- Тип планировщика: плоский;
- Алгоритм планирования: РТ;
- Управление ресурсами: простые семафоры;
- Управление событиями: нет;
- Обработка прерываний: есть, допустимо объявление локальных;
- Максимальное количество задач: 32;
- Максимальное количество ресурсов: 16.

Кроме того, в задание входит написание тестов, проверяющих соответствие проекта этим свойствам.

Введение

ОСРВ включает в себя управляющие модули и набор системных сервисов, доступных пользовательским приложениям. Все объекты ОСРВ создаются статически на этапе компиляции и редактирования связей.

Все приложения, написанные для ОСРВ, удовлетворяющей данной спецификации, должны без ограничений переноситься на любую другую ОСРВ, также удовлетворяющую данной спецификации. Это достигается посредством строгого определения всех системных сервисов, их поведения и ограничений на их использование.

ОСРВ включает в себя подсистемы, подробно рассматриваемые в последующих главах, обеспечивающие следующие возможности:

- Управление задачами;
- Синхронизацию задач посредством использования разделяемых ресурсов.

Основная часть

Управление ресурсами

Когда задача запрашивает ресурсы, мы предоставляем их, если у нее приоритет выше, чем у задачи, которая занимает ресурсы. Если же ресурс свободен, то задача может им воспользоваться.

Содержит следующие процедуры:

- InitResources(int priority) – функция для инициализации ресурса. Осуществляет регистрацию ресурса в системе. Вызывается до работы с ресурсом, чтобы осуществить инициализацию и задать приоритет.
- GetResources(int priority, char* name) – функция для получения ресурса. Содержит проверку на приоритет. То есть, если приоритет у текущей задачи будет ниже, чем у задачи, которая занимает этот ресурс, мы вызываем Interruption. Иначе, мы отдаем этот ресурс, то есть, устанавливаем индекс задачи, которая его занимает и ставим флаг, что ресурс занят.
- ReleaseResource(int index) – функция для освобождения ресурса. Когда задача будет освобождать ресурс, эта функция вызовется и ресурс освободится.
- GetResourceIndex(int priority) – функция для поиска ресурса с заданным приоритетом. Возвращает индекс ресурса из очереди ресурсов с заданным приоритетом.
- ClearResources() – фукнция для очистки ресурсов. Помечаем все индексы ресурсов в очереди -1 и устанавливаем свободные флаги.

Система

Содержит следующие процедуры:

- StartOS(TTaskCall entry, int priority, char* name) – инициализирующая функция. Изначально запущенных задач -1, свободных задач 0, поскольку ничего не запущено. Задаем каждой задаче ссылку на следующую, на последнюю -1. Аналогично поступаем с ресурсами, они все свободны. Осуществляем запуск.
- ShutDownOS() – функция вывода сообщения о выключении ОС.

Задача

Содержит следующие процедуры:

- ActivateTask(TTaskCall entry, int priority, char* name) – функция активации задачи. В очередь задач по индексу помещаем приоритет, имя и то, что она будет делать. Далее запускаем планировщик для этой задачи, чтобы определить после каких задач она будет выполняться.
- TerminateTask(void) – функция для остановки задачи. Если осуществляется попытка завершения задачи, которая уже была завершена, вызывается Interruption. Иначе, останавливаем задачу и освобождаем ресурс.
- Schedule(int index) – функция для определения места выполнения задачи.
- Dispatch(int task) – функция для вызова выполнения задачи.

Прерывания

Содержит следующие структуры:

- struct Interruption {std::string message;} – структура для элемента прерывания.
- class InterruptionHandler – класс обработчик прерываний. Вызывает функцию обработки прерывания, в которой выводится сообщение, которое мы передавали в нашей структуре и выполняется прерывание.

Тесты

Выбор теста осуществляется при помощи конструкции switch-case. Пользователь вводит номер теста, и программа выводит результаты прохождения в консоль.

Доступны следующие тесты:

1. Стресс-тест на запуск максимально возможного количества процессов, по условию задачи – 32.
2. Стресс-тест на захват максимально возможного количества ресурсов, по условию задачи – 17.
3. Проверка на запуск нескольких процессов, задачи с возрастающим приоритетом, согласно алгоритму. Ожидаем, что задачи отработают в порядке согласно их приоритетам.
4. Проверка на запуск двух процессов с захватом ресурса с приоритетом 6. Ожидаем, что первая задача должна отработать с ресурсом и освободить его, затем следующая перехватывает его и успешно с ним отрабатывает.
5. Завершение задачи, которая ранее была завершена. Ожидаем прерывание с сообщением о соответствующей ошибке.

Заключение

В ходе выполнения курсовой работы мною было написана операционная система реального времени, удовлетворяющая свойствам, описанным в индивидуальном задании №3. Стоит отметить, что мною применялись знания, полученные на лекциях и в ходе изучения предоставленных материалов. По завершению проектирования мною был составлен отчет о проделанной работе, в котором представлено описание проекта, а также программный код.

Приложение. Код программы

Файл defs.h:

```
//
//  defs.h
//  Created by Alina Potapova on 22.04.2022.
//

#define MAX_TASK 32
#define MAX_RES 16

```

Файл rtos_api.h:

```
//
//  rtos_api.h
//  Created by Alina Potapova on 22.04.2022.
//

#define DeclareTask(TaskID,priority)\
    TASK(TaskID);                   \
enum {TaskID##prior=priority}

#define DeclareResource(ResID,priority)\
enum {ResID=priority}

#define TASK(TaskID) void TaskID(void)

typedef void TTaskCall(void);

void ActivateTask(TTaskCall entry,int priority,char* name);
void TerminateTask(void);

int StartOS(TTaskCall entry,int priority,char* name);
void ShutdownOS();

void InitResource(int priority);
void GetResource(int priority, char* name);
void ReleaseResource(int priority);
int GetResourceIndex(int priority);
void ClearResources();
```

Файл interruption.hpp:

```
//
//  interruption.hpp
//  Created by Alina Potapova on 22.04.2022.
//

#include <string>

struct Interruption {
    std::string message;
};

class InterruptionHandler {
public:
    InterruptionHandler() = default;
    static void processInterruption(Interruption interruption);
};

```

Файл global.c:

```
//
//  global.c
//  Created by Alina Potapova on 22.04.2022.
//

#include "sys.h"

TTask TaskQueue[MAX_TASK];

TResource ResourceQueue[MAX_RES];

int RunningTask;

int FreeTask;

int FreeResource;

```

Файл sys.h:

```c
//
//  sys.h
//  Created by Alina Potapova on 22.04.2022.
//

#include "defs.h"

#define INSERT_TO_TAIL 1
#define INSERT_TO_HEAD 0

typedef struct Type_Task
{
    int ref;
    int priority;
    int ceiling_priority;
    void (*entry)(void);
    char* name;

} TTask;

typedef struct Type_resource
{
    int index;
    bool isFree;
    int priority;
    char* name;

} TResource;

extern TTask TaskQueue[MAX_TASK];

extern TResource ResourceQueue[MAX_RES];

extern int RunningTask;

extern int FreeTask;

void Schedule(int index);

void Dispatch(int task);

```

Файл os.c:

```c
//
//  os.c
//  Created by Alina Potapova on 22.04.2022.
//

#include <stdio.h>
#include "sys.h"
#include "rtos_api.h"

int StartOS(TTaskCall entry, int priority, char* name)
{
    int i;

    RunningTask = -1;

    FreeTask = 0;

    printf("StartOS!\n");

    for(i = 0; i < MAX_TASK; i++)
    {
        TaskQueue[i].ref = i + 1;
    }

    TaskQueue[MAX_TASK - 1].ref = -1;

    for(i = 0; i < MAX_RES; i++)
    {
        ResourceQueue[i].index = -1;
        ResourceQueue[i].isFree = true;
    }

    ActivateTask(entry, priority, name);

    return 0;
}

void ShutdownOS()
{
    printf("ShutdownOS!\n");
}

```

Файл resource.c:

```
//
//  resource.c
//  Created by Alina Potapova on 22.04.2022.
//

#include "sys.h"
#include "rtos_api.h"
#include "interruption.h"
#include <stdio.h>

void InitResource(int priority) {
    for (int i = 0; i < MAX_RES; i++) {
        if (ResourceQueue[i].priority == 0) {
            ResourceQueue[i].priority = priority;
            break;
        }
    }
}

void GetResource(int priority, char* name)
{
    int task_index;
    int free_occupy;

    for (int i = 0; i < MAX_TASK; i++) {
        if (TaskQueue[i].name == name) {
            task_index = i;
            break;
        }
    }

    if (TaskQueue[RunningTask].priority > TaskQueue[task_index].priority) {
        ClearResources();
        InterruptionHandler::processInterruption({"This process is handled by task with higher/equal priority"});
        return;
    }

    printf("GetResource %s with priority %d\n", name, priority);

    free_occupy = GetResourceIndex(priority);

    if (free_occupy != -1) {
        ResourceQueue[free_occupy].index = RunningTask;
        ResourceQueue[free_occupy].isFree = false;
    } else {
        ClearResources();
        InterruptionHandler::processInterruption({"No free resources with necessary priority is available now"});
    }

}

void ReleaseResource(int task_index)
{
    printf("Release resource %s\n", TaskQueue[task_index].name);

    int index = -1;

    for (int i = 0; i < MAX_RES; i++) {
        if (ResourceQueue[i].index == task_index) {
            index = i;
            break;
        }
    }

    if (index != -1) {
        ResourceQueue[index].index = -1;
        ResourceQueue[index].isFree = true;
    }
}

int GetResourceIndex(int priority) {
    int index = -1;

    for (int i = 0; i < MAX_RES; i++) {
        if (ResourceQueue[i].isFree && ResourceQueue[i].priority >= priority) {
            if (ResourceQueue[i].priority == priority)
                return i;
            if (index == -1 || ResourceQueue[index].priority > ResourceQueue[i].priority)
                index = i;
        }
    }

    return index;
}

void ClearResources() {
    for (int i = 0; i < MAX_RES; i++) {
        ResourceQueue[i].index = -1;
        ResourceQueue[i].isFree = true;
    }
    TerminateTask();
}
```

Файл task.c:

```
//
//  task.c
//  Created by Alina Potapova on 22.04.2022.
//

#include "sys.h"
#include "rtos_api.h"
#include "interruption.h"
#include <setjmp.h>
#include <stdio.h>

void ActivateTask(TTaskCall entry, int priority, char* name)
{
    int task;
    int occupy;

    printf("ActivateTask %s\n", name);

    task = RunningTask;
    occupy = FreeTask;

    FreeTask = TaskQueue[occupy].ref;

    TaskQueue[occupy].priority = priority;
    TaskQueue[occupy].name = name;
    TaskQueue[occupy].entry = entry;

    Schedule(occupy);

    if (task != RunningTask)
    {
        Dispatch(task);
    }

    printf("End of ActivateTask %s\n", name);
}

void TerminateTask(void)
{
    int task;

    task = RunningTask;

    if (task == -1) {
        InterruptionHandler::processInterruption({"Try to terminate not existing task"});
        return;
    }

    printf("Terminate Task %s\n", TaskQueue[task].name);

    RunningTask = TaskQueue[task].ref;

    TaskQueue[task].ref = FreeTask;

    FreeTask = task;

    ReleaseResource(task);

    printf("End of TerminateTask %s\n",TaskQueue[task].name);

}

void Schedule(int index)
{
    int cur;
    int prev;

    printf("Schedule %s\n", TaskQueue[index].name);

    cur = RunningTask;
    prev = -1;

    while(cur != -1 && TaskQueue[cur].priority >= TaskQueue[index].priority)
    {
        prev = cur;
        cur = TaskQueue[cur].ref;
    }

    TaskQueue[index].ref = cur;

    if (prev == -1)
        RunningTask = index;
    else
        TaskQueue[prev].ref = index;

    printf("End of Schedule %s\n", TaskQueue[index].name);

}
```

```
89
90  void Dispatch(int task)
91  {
92      printf("Dispatch\n");
93
94      do
95      {
96          TaskQueue[RunningTask].entry();
97      }
98      while(RunningTask != task);
99
100     printf("End of Dispatch\n");
101
102 }
```

Файл interruption.cpp:

```
1  //
2  //  interruption.cpp
3  //  Created by Alina Potapova on 22.04.2022.
4  //
5
6  #include "interruption.h"
7  #include "rtos_api.h"
8
9  #include <iostream>
10 #include <csignal>
11
12 void InterruptionHandler::processInterruption(Interruption interruption) {
13     std::cout << "Interruption: " << interruption.message << '\n';
14     raise(SIGINT);
15 }
```

Файл test.c:

```
1  //
2  //  test.c
3  //  Created by Alina Potapova on 22.04.2022.
4  //
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <iostream>
9  #include "rtos_api.h"
10
11 DeclareTask(Task1,1);
12 DeclareTask(Task2,2);
13 DeclareTask(Task3,3);
14 DeclareTask(Task4,4);
15 DeclareTask(Task5,4);
16 DeclareTask(Task6,3);
17 DeclareTask(Task7,6);
18 DeclareTask(Task8,7);
19 DeclareTask(Task9,8);
20
21 DeclareResource(Res1,1);
22 DeclareResource(Res2,2);
23 DeclareResource(Res3,3);
24 DeclareResource(Res4,4);
25 DeclareResource(Res5,5);
26 DeclareResource(Res6,6);
27 DeclareResource(Res7,7);
28 DeclareResource(Res8,8);
29 DeclareResource(Res9,9);
30 DeclareResource(Res10,10);
31 DeclareResource(Res11,11);
32 DeclareResource(Res12,12);
33 DeclareResource(Res13,13);
34 DeclareResource(Res14,14);
35 DeclareResource(Res15,15);
36 DeclareResource(Res16,16);
37
38 int main()
39 {
40     int n = 0;
41     std::cout << "Test nember: ";
42     std::cin >> n;
43
44     printf("Hello!\n");
45
46     switch (n) {
47         case 1:
48             StartOS(Task9, Task9prior, (char *)"Task9");
49             break;
50         case 2:
51             for (int i = 0; i < 17; i++) {
52                 InitResource(i);
53             }
54             StartOS(Task7, Task7prior, (char *)"Task7");
55             break;
56         case 3:
57             StartOS(Task1, Task1prior, (char *)"Task1");
58             break;
59         case 4:
60             InitResource(6);
61             StartOS(Task5, Task5prior, (char *)"Task5");
62             break;
63         case 5:
64             StartOS(Task8, Task8prior, (char *)"Task8");
65             TerminateTask();
66         default:
67             break;
```

```
    }
    ShutdownOS();
    return 0;
}


TASK(Task1)
{
    printf("Task1\n");
    ActivateTask(Task2,Task2prior, (char*)"Task2");
    TerminateTask();
}

TASK(Task2)
{
    printf("Task2\n");
    ActivateTask(Task3,Task3prior, (char*)"Task3");
    TerminateTask();
}

TASK(Task3)
{
    printf("Task3\n");
    ActivateTask(Task4,Task4prior, (char*)"Task4");
    TerminateTask();
}

TASK(Task4)
{
    printf("Task4\n");
    TerminateTask();
}

TASK(Task5)
{
    printf("Task5\n");
    GetResource(6, (char*)"Task5");
    ActivateTask(Task6,Task6prior, (char*)"Task6");
    TerminateTask();
}

TASK(Task6)
{
    printf("Task6\n");
    GetResource(6, (char*)"Task6");
    TerminateTask();
}

TASK(Task7)
{
    printf("Task7\n");
    for (int i = 0; i < 16; i++) {
        GetResource(i + 1, (char*)"Task7");
    }
    TerminateTask();
}

TASK(Task8)
{
    printf("Task8\n");
    TerminateTask();
}

TASK(Task9)
{
    printf("Task9\n");
    for(int i = 0; i < 31; i++) {
        ActivateTask(Task4, Task4prior, (char*)"Task4");
    }
    TerminateTask();
}
```