

**ТУ-Варна**

**Дисциплина: Курсов проект по ПОС**

**Проект: Мини ядро за ОС**

**Изготвил: Михаил Терзиев, спец. СИТ, ф.к. 20621672, 3к., 2б гр.**

Задание

1. Да се разработи програмна библиотека на C/C++, реализиращамини ядро на операционна система. Библиотеката да функционира под Linux ОС и да включва следните функционалности:

2. Създаване на задачи;

3. Диспечер за стартиране на задачи в режим на времеделение;

4. За реализацията на задачите и превключването да се използват следните библиотечни функции: makecontext(), getcontext(), swapcontext().

5. За реализацията на диспечера да се разработи системен таймер, генериращ сигнал на всеки 10ms. Да се разработи signal handler, който на базата на тези сигнали да извършва превключването на задачите от опашката на готовите задачи.

6. Всяка задача да се представя чрез контролен блок за управление.

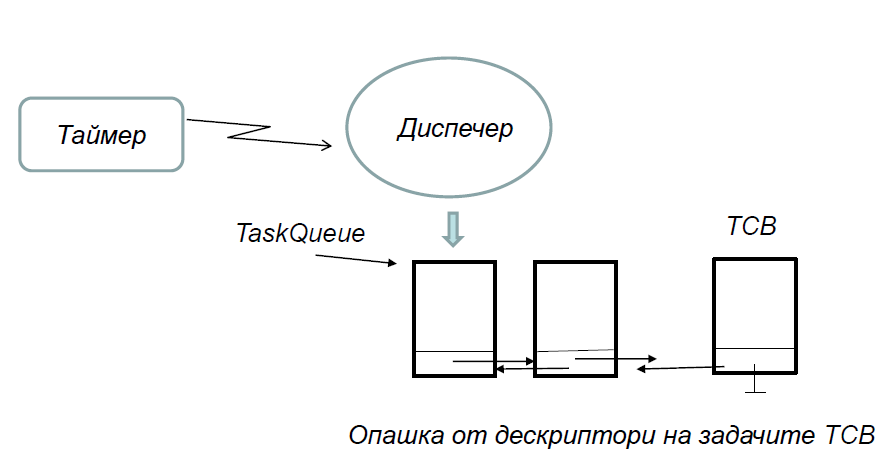
7. Библиотеката да предоставя следния интерфейс за потребителя:

• void \*initlibrary ( ) инициализиране на библиотеката;

• int create\_task ( void \*(\*start\_routine) ) създаване на задача и включването й в опашката на готовите задачи. Аргументът е име на функция, от която се създава задачата. Тази функция не връща резултат и няма параметри. Резултатът от изпълнението на create\_task е 0 при успех и 1 в противен случай.

8. Тестването на библиотеката да се извърши на базата на следния пример програма, стартираща две задачи. Задачите се изпълняват в безкраен цикъл. Едната задача извежда символа плюс на екрана, другата символа минус. На екрана трябва да се наблюдава сменяща се последователност на групи от плюсове и минуси.

9. При предаването на проекта да се представи описание от няколко страници на реализацията му (блок схеми, описание на реализацията, сорс код с коментари в него). Да се представи демонстрация на реализацията.

Структура на системата

Сорс код

task.h

#*ifndef* *TASK\_H*

#*define* *TASK\_H*

#*include* <ucontext.h>

typedef struct tcb\_t

{

ucontext\_t context;

char symbol;

struct tcb\_t\* next;

} tcb\_t;

*extern* tcb\_t task\_queue;

*extern* tcb\_t\* cur\_task;

void *add\_task*(tcb\_t\* new\_task);

tcb\_t\* *get\_first\_task*();

#*endif* // *TASK\_H*

task.c

#*include* <stdlib.h>

#*include* "task.h"

tcb\_t task\_queue;

tcb\_t\* cur\_task;

void *add\_task*(tcb\_t\* new\_task)

{

new\_task->next = *NULL*;

*if* (task\_queue.next == *NULL*)

task\_queue.next = new\_task;

*else*

{

tcb\_t\* last\_task = task\_queue.next;

*while* (last\_task->next != *NULL*)

last\_task = last\_task->next;

last\_task->next = new\_task;

}

}

tcb\_t\* *get\_first\_task*()

{

*if* (task\_queue.next == *NULL*)

*return* *NULL*;

tcb\_t\* next\_task = task\_queue.next;

task\_queue.next = next\_task->next;

next\_task->next = *NULL*;

*return* next\_task;

}

scheduler.h

#*ifndef* *SCHEDULER\_H*

#*define* *SCHEDULER\_H*

void *signal\_dispatcher*(int signum);

void *init\_timer*();

#*endif* // *SCHEDULER\_H*

scheduler.c

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* <signal.h>

#*include* <sys/time.h>

#*include* "scheduler.h"

#*include* "task.h"

void *signal\_dispatcher*(int signum)

{

tcb\_t\* next\_task = *get\_first\_task*();

*if* (next\_task != *NULL*)

{

*add\_task*(cur\_task);

cur\_task = next\_task;

*swapcontext*(&task\_queue.context, &cur\_task->context);

}

}

void *init\_timer*()

{

struct sigaction sa;

struct itimerval timer;

sa.sa\_handler = *signal\_dispatcher*;

*sigemptyset*(&sa.sa\_mask);

sa.sa\_flags = 0;

*sigaction*(*SIGALRM*, &sa, *NULL*);

timer.it\_interval.tv\_sec = 2;

timer.it\_interval.tv\_usec = 0;

timer.it\_value.tv\_sec = 2;

timer.it\_value.tv\_usec = 0;

*setitimer*(*ITIMER\_REAL*, &timer, *NULL*);

}

utils.h

#*ifndef* *UTILS\_H*

#*define* *UTILS\_H*

void\* *initlibrary*();

void *print\_task\_content*();

int *create\_task*(void (\*start\_routine)(), char symbol);

void *loop*();

#*endif* // *UTILS\_H*

utils.c

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* "utils.h"

#*include* "task.h"

#*include* "scheduler.h"

#*define* *FIBER\_STACK* 1024 \* 64

tcb\_t task\_queue;

tcb\_t\* cur\_task;

void\* *initlibrary*()

{

task\_queue.next = *NULL*;

cur\_task = *NULL*;

*init\_timer*();

}

void *print\_task\_content*()

{

*while* (1)

*printf*("%c", cur\_task->symbol);

}

int *create\_task*(void (\*start\_routine)(), char symbol)

{

tcb\_t\* new\_task = (tcb\_t\*)*malloc*(sizeof(tcb\_t));

*if* (new\_task == *NULL*)

*return* -1;

*getcontext*(&new\_task->context);

new\_task->context.uc\_stack.ss\_sp = *malloc*(*FIBER\_STACK*);

new\_task->context.uc\_stack.ss\_size = *FIBER\_STACK*;

new\_task->context.uc\_link = &task\_queue.context;

new\_task->symbol = symbol;

*makecontext*(&new\_task->context, start\_routine, 0);

*add\_task*(new\_task);

*return* 0;

}

void *loop*()

{

cur\_task = *get\_first\_task*();

*while* (1)

{

*if* (cur\_task == *NULL*)

*exit*(0);

tcb\_t\* next\_task = *get\_first\_task*();

*if* (next\_task != *NULL*)

{

*add\_task*(cur\_task);

cur\_task = next\_task;

}

*swapcontext*(&task\_queue.context, &cur\_task->context);

}

}

main.c

#*include* <stdio.h>

#*include* <stdlib.h>

#*include* <signal.h>

#*include* "scheduler.h"

#*include* "utils.h"

int *main*(int argc, char \*argv*[]*)

{

*initlibrary*();

*if* (argc < 2)

{

*printf*("Please provide at least one task!\n");

*return* 1;

}

*for* (int i = 1; i < argc; i++)

{

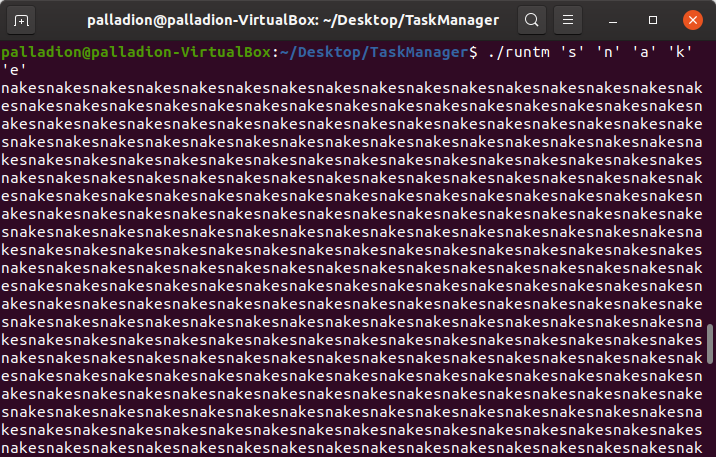
*if* (*create\_task*(*print\_task\_content*, argv[i][0]) != 0)

*return* 1;

}

*loop*();

}

Демонстрация