Технически университет Варна

21621612

Библиотека за мини ядро на операционна система

Тази библиотека реализира мини ядро на операционна система, написано на C/C++. Тя функционира под Linux и предоставя основни функционалности за управление на задачи, включително създаване на задачи и диспечер за времеделение. Библиотеката използва библиотечните функции **makecontext()**, **getcontext()** и **swapcontext()** за управление на контекста на задачите.

**Основни компоненти на библиотеката**

**1. Контролен блок за управление на задачите (Task Control Block - TCB)**

Контролният блок за управление на задачите (TCB) съдържа информация за контекста на задачата, включително стек, идентификатор на задачата и статус. Структурата на TCB е следната:

typedef struct TaskControlBlock {

ucontext\_t context; // Контекст на задачата

struct TaskControlBlock \*next; // Следваща задача в опашката

TCB;

**2. Инициализация на библиотеката**

Функцията **initlibrary** инициализира необходимите структури и настройва системния таймер. Примерна реализация:

void \*initlibrary() {

// Инициализация на опашката на задачите и таймера

// Настройка на сигнал за таймера

}

**3. Създаване на задача**

Функцията **create\_task** създава нова задача и я добавя в опашката на готовите задачи. Примерна реализация:

int create\_task(void \*(\*start\_routine)()) {

TCB \*new\_task = malloc(sizeof(TCB));

if (new\_task == NULL) return 1; // Грешка при алокация

// Настройка на контекста на задачата

getcontext(&new\_task->context);

new\_task->context.uc\_stack.ss\_sp = malloc(STACK\_SIZE);

new\_task->context.uc\_stack.ss\_size = STACK\_SIZE;

new\_task->context.uc\_link = 0;

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

// Добавяне на задачата в опашката

// ...

return 0; // Успех

}

**4. Диспечер на задачите**

Диспечерът превключва между задачите на базата на сигналите от таймера. Примерна реализация:

void scheduler(int signum) {

// Превключване на контекста между задачите

// ...

}

**5. Системен таймер**

Таймерът генерира сигнал на всеки 10ms. Примерна реализация:

void setup\_timer() {

struct sigaction sa;

struct itimerval timer;

sa.sa\_handler = &scheduler;

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = 0;

sigaction(SIGALRM, &sa, NULL);

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

timer.it\_value.tv\_usec = 10000; // 10ms

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

timer.it\_interval.tv\_usec = 10000; // 10ms

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

}

**Примерна програма за тестване**

#include <stdio.h>

#include "minikernel.h" // Заглавен файл на библиотеката

void task1() {

while (1) {

printf("+");

fflush(stdout);

}

}

void task2() {

while (1) {

printf("-");

fflush(stdout);

}

}

int main() {

initlibrary();

create\_task(task1);

create\_task(task2);

// Основен цикъл за управление на задачите

while (1) {

// ...

}

return 0;

}

Тази библиотека предоставя основни функционалности за управление на задачи в Linux. За пълна реализация е необходимо да се добавят механизми за управление на опашката на задачите, обработка на грешки и освобождаване на ресурси. Също така, е важно да се добавят блок-схеми и документация, за да се опише подробно реализацията. Тестовата програма демонстрира основната функционалност на библиотеката, като стартира две задачи, които редуват изхода на символи.