Операционни системи
курсов проект (2)

Реализиране на нишки под Linux

- Под Linux може да се създават нишки, които се изпълняват в потребителски режим – fibers.
- Работата с fibers се извършва чрез библиотечни функции, представени в **`ucontext.h`**.
- Управлението на fibers е базирано на манипулиране на контекста им на изпълнение.

Библиотечни функции

- **`int getcontext(ucontext_t *ucp)`** – съхранява текущия контекст на изпълнение в аргумента
- **`int setcontext(const ucontext_t *ucp)`** – превключва към специфичен контекст на изпълнение
- **`void makecontext(ucontext_t *ucp, void (*func)(), int argc, ...)`** – модифицира указан контекст на изпълнение. При активиране на този контекст се извиква функцията и й се предават аргументите
- **`int swapcontext(ucontext_t *oucp, const ucontext_t *ucp)`** – съхранява текущия контекст в **`oucp`** и активира контекст на изпълнение - **`ucp`**

Системни типове данни

```
struct ucontext_t {
  ucontext_t *uc_link;   // указател към контекста,
   който ще се възстанови ако се върнем от контекста, описан
   в текущата структура
  sigset_t  uc_sigmask; // множество от сигнали, които
  се блокират, когато този контекст е активен
  stack_t  uc_stack;     // стекът, използван от
  контекста
  mcontext_t  uc_mcontext; // специфично за
  машината представяне на съхранения контекст
 ...
}
```

Системни типове данни

```
struct stack_t {
  void     *ss_sp;     // базовия адрес на стека
  size_t    ss_size;   // размер на стека
  int       ss_flags;  // флагове
}
```

Създаване на fiber

1. Извиква се **`getcontext`** за получаване на текущия контекст на изпълнение.
2. Модифицират се членовете на структурата **`ucontext_t`** да указват вече новия контекст. Заема се памет за новия стек.
3. Извиква се **`makecontext`** и се указват функцията от която ще се изпълнява fiber и нейните аргументи.

Създаване на fiber

```
#define FIBER_STACK 1024*64
ucontext_t uctx_func, utcx_main;

void func1 () {
       printf("Hello I am func1\n");
}

int main() {
  getcontext(&uctx_func);
  uctx_func.uc_link = NULL;
  uctx_func.uc_stack.ss_sp = malloc(FIBER_STACK);
  uctx_func.uc_stack.ss_size = FIBER_STACK;
  uctx_func.uc_stack.ss_flags = 0;
  makecontext(&uctx_func, func1, 0);
  ...
}
```

Превключване на контекст

```
#define FIBER_STACK 1024*64
ucontext_t uctx_func, uctx_main;

void func1 () {
        printf("Hello I am func1\n");
}

int main() {
  getcontext(&uctx_func);
  uctx_func.uc_link = NULL;
  uctx_func.uc_stack.ss_sp = malloc(FIBER_STACK);
  uctx_func.uc_stack.ss_size = FIBER_STACK;
  uctx_func.uc_stack.ss_flags = 0;
  makecontext(&uctx_func, func1, 0);

  swapcontext(&uctx_main, &uctx_func);
  ...
}
```

Пример за превключване на контекст

Да се създаде fiber. Да се реализира изпълнението в последователността:
нишка -> main() -> нишка -> main ()

Преди всяко превключване да изведат съобщения.

```
#include <malloc.h>
#include <ucontext.h>
#include <stdio.h>

#define FIBER_STACK 1024*64
ucontext_t uctx_func, uctx_main;

void fiber () {
  printf("Fiber: started\n");
  printf("Fiber: Now swap context to main\n");
  swapcontext(&uctx_func, &uctx_main);
  printf("Fiber: returned\n");
  printf("Fiber: Now swap context again to main\n");
  swapcontext(&uctx_func, &uctx_main);
}
```

Пример за превключване на контекст

```
int main() {
  getcontext(&uctx_func);
  uctx_func.uc_link = NULL; // ако се укаже &uctx_main, то при завършване на
// функцията, за която е създаден контекста, ще се активира този контекст. В
// този случай не е необходимо накрая на функцията да се превключва контекста
// към main()
  uctx_func.uc_stack.ss_sp = malloc(FIBER_STACK);
  uctx_func.uc_stack.ss_size = FIBER_STACK;
  uctx_func.uc_stack.ss_flags = 0;
  makecontext(&uctx_func, fiber, 0);

  printf("Main: swap context to fiber\n");
  swapcontext(&uctx_main, &uctx_func);

  printf("Main: returned\n");
  printf("Main: swap context again to fiber\n");
  swapcontext(&uctx_main, &uctx_func);

  printf("Main: returned\n");
  printf("Main: exiting\n");

  return 0;
}
```

Задачи за изпълнение

1. Да се реализира примера.
2. Да се реализира примера с коментара.
3. Да се създадат 3 fiber-а. Да се реализира изпълнението в последователността:

   fiber1 -> fiber2-> fiber3 -> fiber1 -> fiber2 -> fiber3 -> main()

Да се изведат подходящи съобщения за проверка коректността на изпълнение.