LAB1 : User level thread

By qqc

qiuqichen@bupt.edu.cn

User level thread

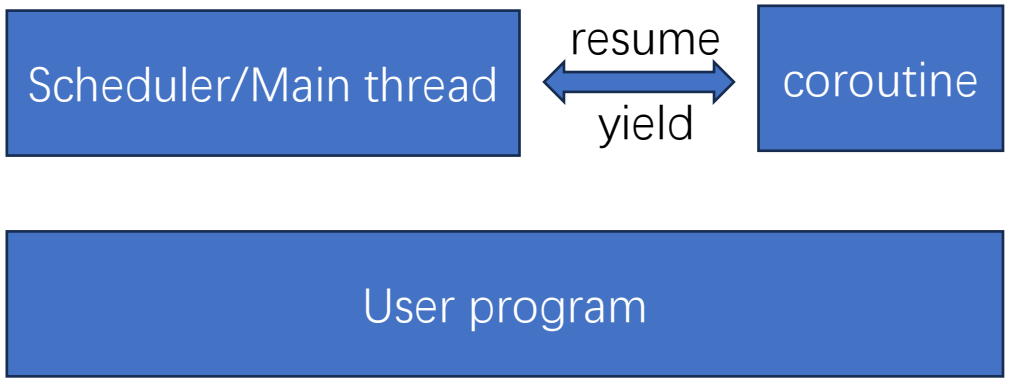
- Background
- Setup the environment
- Brief introduction to skeleton of code
- Debug
- Submitting

User level thread

- Background
- Setup the environment
- Brief introduction to skeleton of code
- Debug
- Submitting

Basic idea

- We are going to implement a simple stackful coroutine.
  - Asymmetric means there is a scheduler.
  - Stackful means each instance of a coroutine has its own stack.
    - In contrast to stackless coroutines, stackful coroutines allow invoking the suspend operation out of arbitrary sub-stackframes, enabling escape-and-reenter recursive operations.
- Here's a diagram of our design.



Context

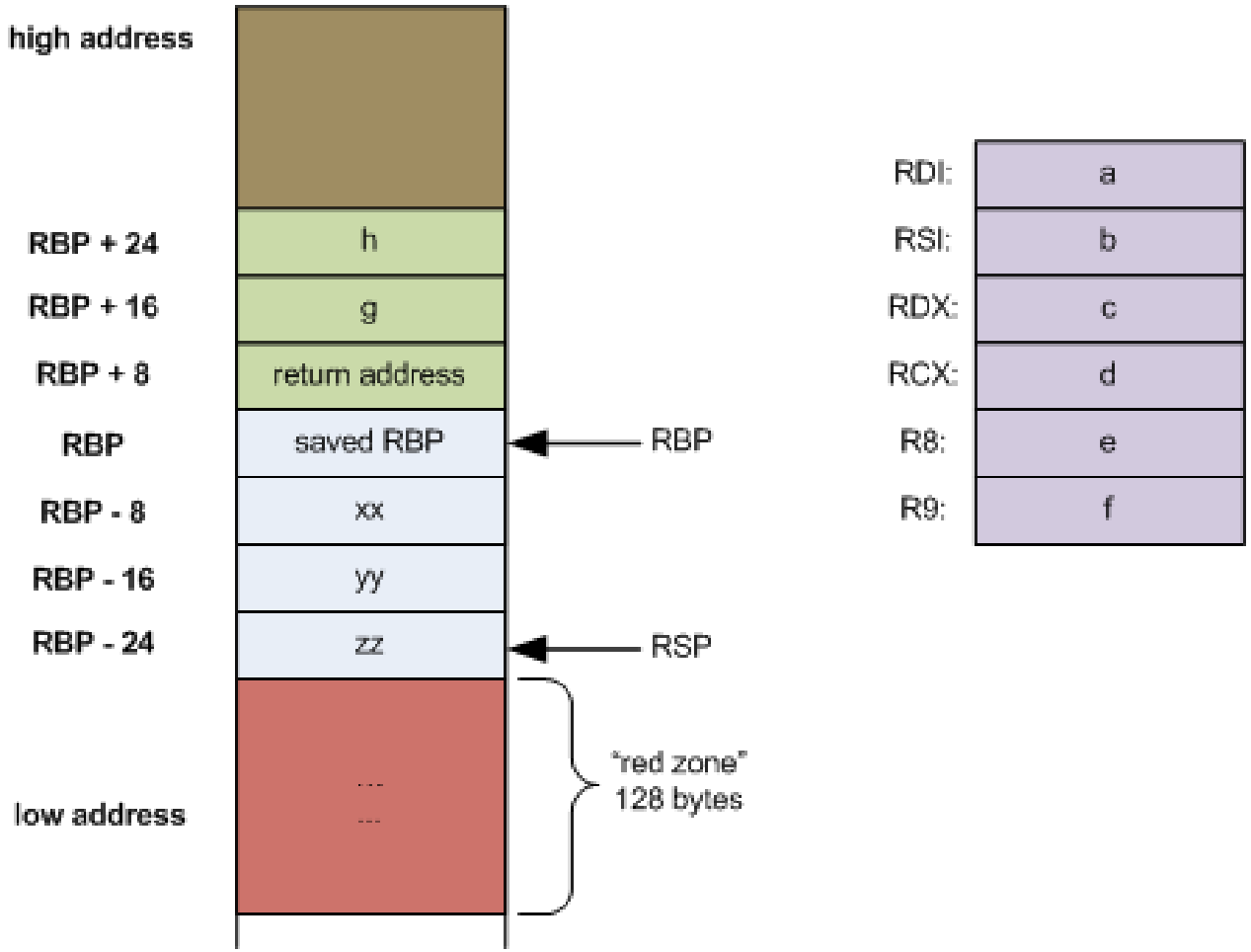
- A context is a data structure that stores some key information when executing a user-level thread.
  - No need to store temporary variables or heap variables
    - But we do need some additional space to make a stack for each coroutine.
  - Store a subset of registers is enough.
  - FPU and some flag registers are not considered.

8-byte register	Bytes 0-3	Bytes 0-1	Byte 0
%rax	%eax	%ax	%al
%rcx	%ecx	%cx	%cl
%rdx	%edx	%dx	%dl
%rbx	%ebx	%bx	%bl
%rsi	%esi	%si	%sil
%rdi	%edi	%di	%dil
%rsp	%esp	%sp	%spl
%rbp	%ebp	%bp	%bpl
%r8	%r8d	%r8w	%r8b
%r9	%r9d	%r9w	%r9b
%r10	%r10d	%r10w	%r10b
%r11	%r11d	%r11w	%r11b
%r12	%r12d	%r12w	%r12b
%r13	%r13d	%r13w	%r13b
%r14	%r14d	%r14w	%r14b
%r15	%r15d	%r15w	%r15b

Register	Purpose	Saved across calls
%rax	temp register; return value	No
%rbx	callee-saved	Yes
%rcx	used to pass 4th argument to functions	No
%rdx	used to pass 3rd argument to functions	No
%rsp	stack pointer	Yes
%rbp	callee-saved; base pointer	Yes
%rsi	used to pass 2nd argument to functions	No
%rdi	used to pass 1st argument to functions	No
%r8	used to pass 5th argument to functions	No
%r9	used to pass 6th argument to functions	No
%r10-r11	temporary	No
%r12-r15	callee-saved registers	Yes

Calling convention

```
long myfunc(long a, long b, long c, long d,
            long e, long f, long g, long h)
{
    long xx = a * b * c * d * e * f * g * h;
    long yy = a + b + c + d + e + f + g + h;
    long zz = utilfunc(xx, yy, xx % yy);
    return zz + 20;
}
```



Assembly

- Use ChatGPT to help you read or write (if needed) the assembly.

这段汇编代码看起来是一个线程上下文切换的函数。当一个线程在执行中被打断，为了将控制转移到另一个线程，你需要保存当前线程的状态并恢复另一个线程的状态。

让我们逐条解读这段代码：

1. `pop %rax`：从栈顶取值放入rax寄存器中，这通常是调用者在调用线程切换函数之前将return address push到栈上的地址。

2-9. `movq %rXX, offset(%rdi)`：保存当前线程的寄存器值。rdi中通常存储当前线程的上下文或结构体地址。

10-17. `movq offset(%rsi), %rXX`：恢复新线程的寄存器值。rsi中通常存储的是新线程的上下文或结构体地址。

1. `push %rax`：将rax中的值（先前由`pop %rax`取得的返回地址）推入栈中。

19-22. `movq offset(%r10), %rXX`：从r10寄存器指向的结构体或数组中恢复额外的寄存器值。

1. `ret`：从栈顶取return address并返回，这实际上会跳转到新线程的执行位置。

注意：

- `rdi`和`rsi`是x86-64调用约定中的前两个参数传递寄存器，所以在这里，我们可以推测`rdi`指向的是当前线程的上下文，而`rsi`指向的是新线程的上下文。
- `%r10`似乎被用作临时存储，可能在外部的代码中设置了某种上下文结构的地址，里面包含了其他的寄存器状态。

总之，这段代码保存了当前线程的状态，恢复了新线程的状态，然后通过`ret`指令将控制权交给了新线程。

Regen

User level thread

- Background
- Setup the environment
- Brief introduction to skeleton of code
- Debug
- Submitting

Setup the environment

- You can use your linux environment directly
  - gcc , git , make are needed
- If you’re using a dev container in Windows, you may encounter some problems:
  - You may need to install wsl2 in Win11
    - wsl --update
  - Downgrade dev container to 0.26.1
  - Sometimes it may fail to fetch software when building due to network timeout. Just try again.

- Any question?

User level thread

- Background
- Setup the environment
- Brief introduction to the skeleton of code
- Debug
- Submitting

Overview

```
.
|-- Makefile
|-- README.md
|-- pingpong.c   test1
|-- recursion.c    test2
|-- simple.c       test3
|-- switch.S        assembly that used to exchange context
|-- uthread.c      user thread library(your work)
`-- uthread.h      declaration of user thread library
```

Example Usage of thread_switch

```
int main(){
    long long sp;
    main_thread = malloc(sizeof(struct uthread));
    memset(main_thread,0,sizeof(struct uthread));

    current_thread = malloc(sizeof(struct uthread));
    memset(current_thread,0,sizeof(struct uthread));

    current_thread->context.rip = (long long)test;
    sp = ((long long)&current_thread->stack + STACK_SIZE) & (~(long long)15);
    sp -= 8;
    *(long long*)sp = (long long)bug;
    current_thread->context.rsp = sp;

    thread_switch(&main_thread->context,&current_thread->context);
    printf("main\n");
    return 0;
}
```

Example(cont.)

```
void bug(){
    printf("missing return address\n");
    exit(-1);
}

void test(){
    printf("test\n");
    thread_switch(&current_thread->context,&main_thread->context);
}
```

Other tips

- Put function entry as `_uthread_entry` so that you can pass an argument and set the thread flag
- There are many reasons why you encounter segment fault(for example, setting a wrong pointer). You can use gdb or print to locate the wrong.

User level thread

- Background
- Setup the environment
- Brief introduction to the skeleton of code
- Debug
- Submitting

A introduction to gdb

Search “gdb cheat sheet” or ask ChatGPT
e.g: GDB cheat sheet (github.com)

command	function
r or run	run the program
c or continue	continue
b or break	Set breakpoint
info locals	See local variables
info reg	See current register value
info args	See current arguments
p or print	Print data
x	Print data in the address
n , ni	next line, next assembly line
s , si	step(do not skip function), step a assembly line
layout	Change layout of gdb(option: layout tui,layout asm,layout src)
focus	Focus on command line or window
frame	See current frame
bt	Back trace. See previous frame

Example

- gdb ./demo

Other useful debugging tools

- objdump
- print(Remember to flush cache)
- ChatGPT
- etc.

User level thread

- Background
- Setup the environment
- Brief introduction to the skeleton of code
- Debug
- Submitting

Submitting

- git diff –color [commit id] > foo.txt
  - E.g: git diff --color 5bd4126 > example.patch

- If you want to verify your patch:
  - git checkout 5bd4126
  - git apply example.patch
    - check if all your changes are applied.

Q&A

Feel free to ask questions.