ucontext 原理

jask

2024-08-26

ucontext 函数

```
int getcontext(ucontext_t *ucp);
int setcontext(const ucontext_t *ucp);
void makecontext(ucontext_t *ucp, void (*func)(), int argc, ...);
int swapcontext(ucontext_t *oucp, ucontext_t *ucp);
```

具体功能：getcontext 获取线程的当前上下文；setcontext 相反是从 ucp 中恢复出上下文；makecontext 是修改 ucp 指向的上下文环境，swapcontext 是保存当前上下文，并切换到新的上下文。下面看他们的具体实现：

具体实现

```
ENTRY(__getcontext)
    /* Save the preserved registers, the registers used for passing
       args, and the return address.  */
    movq    %rbx, oRBX(%rdi)
    movq    %rbp, oRBP(%rdi)
    movq    %r12, oR12(%rdi)
    movq    %r13, oR13(%rdi)
    movq    %r14, oR14(%rdi)
    movq    %r15, oR15(%rdi)

    movq    %rdi, oRDI(%rdi)
    movq    %rsi, oRSI(%rdi)
    movq    %rdx, oRDX(%rdi)
    movq    %rcx, oRCX(%rdi)
    movq    %r8, oR8(%rdi)
    movq    %r9, oR9(%rdi)

    movq    (%rsp), %rcx
    movq    %rcx, oRIP(%rdi)
    leaq    8(%rsp), %rcx       /* Exclude the return address.  */
    movq    %rcx, oRSP(%rdi)

    /* We have separate floating-point register content memory on the
       stack.  We use the __fpregs_mem block in the context.  Set the
       links up correctly.  */

    leaq    oFPREGSMEM(%rdi), %rcx
    movq    %rcx, oFPREGS(%rdi)
    /* Save the floating-point environment.  */
    fnstenv (%rcx)
    fldenv  (%rcx)
    stmxcsr oMXCSR(%rdi)

    /* Save the current signal mask with
       rt_sigprocmask (SIG_BLOCK, NULL, set,_NSIG/8).  */
    leaq    oSIGMASK(%rdi), %rdx
    xorl    %esi,%esi
#if SIG_BLOCK == 0
```

```
    xorl    %edi, %edi
#else
    movl    $SIG_BLOCK, %edi
#endif
    movl    $_NSIG8,%r10d
    movl    $__NR_rt_sigprocmask, %eax
    syscall
    cmpq    $-4095, %rax        /* Check %rax for error.  */
    jae SYSCALL_ERROR_LABEL /* Jump to error handler if error.  */

    /* All done, return 0 for success.  */
    xorl    %eax, %eax
    ret
PSEUDO_END(__getcontext)
```

getcontext 的汇编代码中，第一部分就是保存当前上下文中的各个寄存器到第一个参数 rdi 中，即 ucontext_t 中，其中目标操作数 (%rdi) 前面的 oRBX，oRBP…的含义如下，

```
#define ucontext(member)    offsetof (ucontext_t, member)
#define mcontext(member)    ucontext (uc_mcontext.member)
#define mreg(reg)       mcontext (gregs[REG_##reg])

oRBP        mreg (RBP)
oRSP        mreg (RSP)
oRBX        mreg (RBX)
oR8     mreg (R8)
oR9     mreg (R9)
oR10        mreg (R10)
oR11        mreg (R11)
oR12        mreg (R12)
oR13        mreg (R13)
oR14        mreg (R14)
```

进入 getcontext 之后

首先保存rbx，rbp，r12，r13，r14，r15，这6个数据寄存器，因为他们遵循被调用者使用，所以需要保存，
然后是保存rdi，rsi，rdx，rcx，r8，r9这6个寄存器，因为它用于保存函数参数，也是遵循被调用者使用。但大家发现没有，g
其次，读取rsp寄存器指向的进程stack栈顶中的RIP值，该栈顶的值，是在调用getcontext时，即执行call指令时，默认会做的事
再次，将栈顶指针加8，即获得调用getcontext()之前的栈顶指定，并保存到ucontext中，当恢复时，恢复到RSP寄存器中。

getcontext 的第二部分设置浮点计数器，第三部分就是保存当前线程的信号屏蔽掩码；

makecontext 实现

```
void
__makecontext (ucontext_t *ucp, void (*func) (void), int argc, ...)
{
  extern void __start_context (void);
  greg_t *sp;
  unsigned int idx_uc_link;
  va_list ap;
  int i;

  /* Generate room on stack for parameter if needed and uc_link.  */
  sp = (greg_t *) ((uintptr_t) ucp->uc_stack.ss_sp
           + ucp->uc_stack.ss_size);
  sp -= (argc > 6 ? argc - 6 : 0) + 1;
  /* Align stack and make space for trampoline address.  */
  sp = (greg_t *) ((((uintptr_t) sp) & -16L) - 8);

  idx_uc_link = (argc > 6 ? argc - 6 : 0) + 1;

  /* Setup context ucp.  */
```

```
  /* Address to jump to.  */
  ucp->uc_mcontext.gregs[REG_RIP] = (uintptr_t) func;
  /* Setup rbx.*/
  ucp->uc_mcontext.gregs[REG_RBX] = (uintptr_t) &sp[idx_uc_link];
  ucp->uc_mcontext.gregs[REG_RSP] = (uintptr_t) sp;

  /* Setup stack.  */
  sp[0] = (uintptr_t) &__start_context;
  sp[idx_uc_link] = (uintptr_t) ucp->uc_link;

  va_start (ap, argc);
  /* Handle arguments.

     The standard says the parameters must all be int values.  This is
     an historic accident and would be done differently today.  For
     x86-64 all integer values are passed as 64-bit values and
     therefore extending the API to copy 64-bit values instead of
     32-bit ints makes sense.  It does not break existing
     functionality and it does not violate the standard which says
     that passing non-int values means undefined behavior.  */
  for (i = 0; i < argc; ++i)
    switch (i)
      {
      case 0:
    ucp->uc_mcontext.gregs[REG_RDI] = va_arg (ap, greg_t);
    break;
      case 1:
    ucp->uc_mcontext.gregs[REG_RSI] = va_arg (ap, greg_t);
    break;
      case 2:
    ucp->uc_mcontext.gregs[REG_RDX] = va_arg (ap, greg_t);
    break;
      case 3:
    ucp->uc_mcontext.gregs[REG_RCX] = va_arg (ap, greg_t);
    break;
      case 4:
    ucp->uc_mcontext.gregs[REG_R8] = va_arg (ap, greg_t);
    break;
      case 5:
    ucp->uc_mcontext.gregs[REG_R9] = va_arg (ap, greg_t);
    break;
      default:
    /* Put value on stack.  */
    sp[i - 5] = va_arg (ap, greg_t);
    break;
      }
  va_end (ap);
}
```

makecontext 用于修改已经获取的上下文信息，其支持将运行 stack 切换为用户自定义栈，并可以修改 ucontext 上下文中保存的 RIP 指针，这样当恢复此 ucontext 的上下文时，就会将 RIP 寄存器的恢复为 ucontext 中的 RIP 字段值，跳到指定的代码处进行执行，这也是协程运行的基本要求。makecontex 的 glibc 实现中，

首先是对用户自定义栈进行处理，将sp移动到栈底(栈空间是递减的)，然后进行对齐，并预留出8字节的trampoline空间(防止相
然后，将传入的上下文ucontext中的rip字段设置为fun函数的地址，rbx字段指向继承上下文，rsp字段指向自定义栈的栈顶
其次就是将start_context和uc_link，存入栈中
最后，是将makecontext的参数存入ucontext的上下文中，对于多余的参数，进行压栈操作

swapcontext

```
ENTRY(__swapcontext)
    /* Save the preserved registers, the registers used for passing args,
```

```
   and the return address.  */
movq    %rbx, oRBX(%rdi)
movq    %rbp, oRBP(%rdi)
movq    %r12, oR12(%rdi)
movq    %r13, oR13(%rdi)
movq    %r14, oR14(%rdi)
movq    %r15, oR15(%rdi)

movq    %rdi, oRDI(%rdi)
movq    %rsi, oRSI(%rdi)
movq    %rdx, oRDX(%rdi)
movq    %rcx, oRCX(%rdi)
movq    %r8, oR8(%rdi)
movq    %r9, oR9(%rdi)

movq    (%rsp), %rcx
movq    %rcx, oRIP(%rdi)
leaq    8(%rsp), %rcx       /* Exclude the return address.  */
movq    %rcx, oRSP(%rdi)

/* We have separate floating-point register content memory on the
   stack.  We use the __fpregs_mem block in the context.  Set the
   links up correctly.  */
leaq    oFPREGSMEM(%rdi), %rcx
movq    %rcx, oFPREGS(%rdi)
/* Save the floating-point environment.  */
fnstenv (%rcx)
stmxcsr oMXCSR(%rdi)


/* The syscall destroys some registers, save them.  */
movq    %rsi, %r12

/* Save the current signal mask and install the new one with
   rt_sigprocmask (SIG_BLOCK, newset, oldset,_NSIG/8).  */
leaq    oSIGMASK(%rdi), %rdx
leaq    oSIGMASK(%rsi), %rsi
movl    $SIG_SETMASK, %edi
movl    $_NSIG8,%r10d
movl    $__NR_rt_sigprocmask, %eax
syscall
cmpq    $-4095, %rax        /* Check %rax for error.  */
jae SYSCALL_ERROR_LABEL /* Jump to error handler if error.  */

/* Restore destroyed registers.  */
movq    %r12, %rsi

/* Restore the floating-point context.  Not the registers, only the
   rest.  */
movq    oFPREGS(%rsi), %rcx
fldenv  (%rcx)
ldmxcsr oMXCSR(%rsi)

/* Load the new stack pointer and the preserved registers.  */
movq    oRSP(%rsi), %rsp
movq    oRBX(%rsi), %rbx
movq    oRBP(%rsi), %rbp
movq    oR12(%rsi), %r12
movq    oR13(%rsi), %r13
movq    oR14(%rsi), %r14
movq    oR15(%rsi), %r15
```

```
    /* The following ret should return to the address set with
    getcontext.  Therefore push the address on the stack.  */
    movq    oRIP(%rsi), %rcx
    pushq   %rcx

    /* Setup registers used for passing args.  */
    movq    oRDI(%rsi), %rdi
    movq    oRDX(%rsi), %rdx
    movq    oRCX(%rsi), %rcx
    movq    oR8(%rsi), %r8
    movq    oR9(%rsi), %r9

    /* Setup finally  %rsi.  */
    movq    oRSI(%rsi), %rsi

    /* Clear rax to indicate success.  */
    xorl    %eax, %eax
    ret
PSEUDO_END(__swapcontext)
```

示例

```
#include <stdlib.h>
#include <ucontext.h>
#include <stdio.h>
#include <string.h>

ucontext_t uc, ucm;

void foo()
{
    printf("%s\n", __FUNCTION__);
}

int main()
{
    // allocate stack
    size_t co_stack_size = 64*1024;
    char * co_stack = (char *)malloc(co_stack_size);
    memset(co_stack, 0, co_stack_size);

    //get current context
    getcontext(&uc);

    // make ucontext to run foo
    uc.uc_stack.ss_sp = co_stack;
    uc.uc_stack.ss_size = co_stack_size;
    uc.uc_link = &ucm;
    makecontext(&uc, &foo, 0);

    // switching back-and-forth for 100 times
    for (int i = 0; i < 100; i++)
    {
       swapcontext(&ucm, &uc);
    }

    free(co_stack);
    return 0;
}
```