华东师范大学数据科学与工程学院实验报告

课程名称：操作系统	**年级**：2020 级	**上机实践成绩**：
指导教师：翁楚良	**姓名**：张熙翔	**学号**：10205501427
上机实践名称：EDF 近似实时调度		**上机实践日期**：2022/4/18

一、实验目的

1. 巩固操作系统的进程调度机制和策略
2. 熟悉 MINIX 系统调用和 MINIX 调度器的实现

二、实验任务

在 MINIX3 中实现 Earliest-Deadline-First 近似实时调度功能：

1. 提供设置进程执行期限的系统调度 chrt (long deadline)，用于将调用该系统调用的进程设为实时进程，其执行的期限为：从调用处开始 deadline 秒。

2. 在内核进程表中需要增加一个条目，用于表示进程的实时属性；修改相关代码，新增一个系统调用 chrt，用于设置其进程表中的实时属性。

3. 修改 proc.c 和 proc.h 中相关的调度代码,实现最早 deadline 的用户进程相对于其它用户进程具有更高的优先级，从而被优先调度运行。

4. 在用户程序中，可以在不同位置调用多次 chrt 系统调用，在未到 deadline 之前，调用 chrt 将会改变该程序的 deadline。

5. 未调用 chrt 的程序将以普通的用户进程(非实时进程)在系统中运行。

三、使用环境

物理机：Windows10
虚拟机：Minix3
虚拟机软件：Vmware
代码编辑：VScode
物理机与虚拟机文件传输：FileZilla

四、实验过程

MINIX3 中的系统调用结构分成三个层次：应用层，服务层，内核层。在这三层中分别进行代码修改，实现系统调用 chrt 的信息传递。从应用层用_syscall 将信息传递到服务层，在服务层用_kernel_call 将信息传递到内核层，在内核层对进程结构体增加 deadline 成员。

应用层：

1. 在/usr/src/include/unistd.h 中添加 chrt 函数定义

```
int chrt(long );//添加 chrt 函数定义
```

2. 在/usr/src/minix/lib/libc/sys/chrt.c 中添加 chrt 函数实现

```
#include <lib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include<sys/time.h>

int chrt(long deadline){
    //struct timespec time={0,0};
    struct  timeval    tv;
    struct  timezone   tz;
    message m;
    memset(&m,0,sizeof(m));
    //设置 alarm
    alarm((unsigned int)deadline);
    //将当前时间记录下来 算 deadline
    if(deadline>0){
        gettimeofday(&tv,&tz);
        deadline = tv.tv_sec + deadline;
    }
    //存 deadline
m.m2_l1=deadline;
return(_syscall(PM_PROC_NR,PM_CHRT,&m));
}
```

3. 在/usr/src/minix/lib/libc/sys 中 Makefile.inc 文件添加 chrt.c 条目

```
 environ.c __getcwd.c vfork.c
it.c setpgid.c chrt.c
```

服务层：

1. 在/usr/src/minix/servers/pm/proto.h 中添加 do_chrt 函数定义。

```
int do_fork(void);
int do_srv_fork(void);
int do_chrt(void);//添加 chrt 函数定义
```

2. 在/usr/src/minix/servers/pm/chrt.c 中添加 do_chrt 函数实现，调用 sys_chrt()

```
int do_chrt(){
    sys_chrt(who_p,m_in.m2_l1);
    return (OK);
}
```

3. 在/usr/src/minix/servers/pm/Makefile 中添加 chrt.c 条目。

```
y.c table.c trace.c getset.c m
ext.c schedule.c chrt.c
```

4. 在/usr/src/minix/include/minix/callnr.h 中定义 PM_CHRT 编号。

```
#define PM_CHRT         (PM_BASE + 48)
#define NR_PM_CALLS     49  /* highest number from base plus one */
```

5. 在/usr/src/minix/servers/pm/table.c 中调用映射表。

```
    CALL(PM_EXIT)       = do_exit,      /* _exit(2) */
    CALL(PM_FORK)       = do_fork,      /* fork(2) */
    CALL(PM_CHRT)           = do_chrt//参照 fork、exit 定义进行添加。
```

6. 在/usr/src/minix/include/minix/syslib.h 中添加 sys_ chrt () 定义。

```
int sys_chrt(endpoint_t proc_ep,long deadline);
```

7. 在/usr/src/minix/lib/libsys/sys_chrt.c 中添加 sys_chrt () 实现。

```
#include"syslib.h"
int sys_chrt(proc_ep,deadline)
endpoint_t proc_ep;
long deadline;
{
int r;
message m;
//将进程号和 deadline 放入消息结构体
m.m2_i1=proc_ep;
m.m2_l1=deadline;
```

```
//通过_kernel_call 传递到内核层
r=_kernel_call(SYS_CHRT,&m);
return r;
}
//参照该文件夹下的 sys_fork 文件，在实现中通过_kernel_call (调用号)向内核传递。
```

8. 在/usr/src/minix/lib/libsys 中的 Makefile 中添加 sys_chrt.c 条目。

```
sys_clear.c \
sys_chrt.c \
sys_cprof.c \
```

内核层：

1. 在/usr/src/minix/kernel/system.h 中添加 do_chrt 函数定义。

```
int do_fork(struct proc * caller, message *m_ptr);
#if ! USE_FORK
#define do_fork NULL
#endif

int do_chrt(struct proc * caller, message *m_ptr);
#if ! USE_CHRT
#define do_chrt NULL
#endif
```

2. 在/usr/src/minix/kernel/system/do_ chrt.c 中添加 do_chrt 函数实现。

```
#include "kernel/system.h"
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <lib.h>
#include <minix/endpoint.h>
#include <string.h>

int do_chrt(struct proc *caller, message *m_ptr)
{
  struct proc *rp;
  long exp_time;

  exp_time = m_ptr->m2_l1;

//通过 proc_addr 定位内核中进程地址
```

```
  rp = proc_addr(m_ptr->m2_i1);
//将 exp_time 赋值给该进程的 p_deadline
  rp->p_deadline = exp_time;

  return (OK);
}
```

3. 在/usr/src/minix/kernel/system/ 中 Makefile.inc 文件添加 do_chrt.c 条目。

4. 在/usr/src/minix/include/minix/com.h 中定义 SYS_CHRT 编号以及总数。

```
#  define SYS_CHRT (KERNEL_CALL + 58)
/* Total */
#define NR_SYS_CALLS    59  /* number of kernel calls */
```

5. 在/usr/src/minix/kernel/system.c 中添加 SYS_CHRT 编号到 do_chrt 的映射。

```
 map(SYS_FORK, do_fork);     /* a process forked a new process */
  map(SYS_EXEC, do_exec);   /* update process after execute */
  map(SYS_CHRT,do_chrt);//参考 do_fork、do_exec 按照相应格式添加。
```

6. 在/usr/src/minix/commands/service/parse.c 的 system_tab 中添加名称编号对。

```
 { "CHRT",    SYS_CHRT },
```

进程调度模块位于/usr/src/minix/kernel/下的 proc.h 和 proc.c，修改影响进程调度顺序的部分：

/usr/src/minix/kernel/下的 proc.h 中

```
  long long p_deadline;//设置 deadline
```

/usr/src/minix/kernel/下的 proc.c

```
 if (rp->p_deadline >0)
    {
          rp->p_priority = 5;
    }
```

```
rdy_head = get_cpulocal_var(run_q_head);
  for (q=0; q < NR_SCHED_QUEUES; q++) {
```

```
      //优先级队列为空时
    if(!(rp = rdy_head[q])) {
        TRACE(VF_PICKPROC, printf("cpu %d queue %d empty\n", cpuid, q););
        continue;
    }
          rp=rdy_head[q];
          temp=rp->p_nextready;
       if(q==5){
          while(temp!=NULL){
              if (temp->p_deadline > 0)
              {
                  if (rp->p_deadline == 0|| (temp->p_deadline <
rp->p_deadline))
                  {
                      if (proc_is_runnable(temp))
                          rp = temp;
                  }
              }
              temp = temp->p_nextready;
          }
      }
```

五、实验结果

```
# ./test
proc1 set success
proc2 set success
proc3 set success
# prc3 heart beat 1
prc2 heart beat 1
prc1 heart beat 1
prc3 heart beat 2
prc2 heart beat 2
prc1 heart beat 2
prc3 heart beat 3
prc2 heart beat 3
prc1 heart beat 3
prc3 heart beat 4
prc2 heart beat 4
prc1 heart beat 4
prc3 heart beat 5
Change proc1 deadline to 5s
prc1 heart beat 5
prc2 heart beat 5
prc3 heart beat 6
prc1 heart beat 6
prc2 heart beat 6
prc3 heart beat 7
prc1 heart beat 7
prc2 heart beat 7
prc3 heart beat 8
prc1 heart beat 8
prc2 heart beat 8
prc3 heart beat 9
Change proc3 deadline to 3s
prc3 heart beat 10
prc2 heart beat 9
prc3 heart beat 11
prc2 heart beat 10
prc2 heart beat 11
prc2 heart beat 12
prc2 heart beat 13
```