《操作系统》实验

二、进程管理

助教：彭小双、于春琳
指导教师：翁楚良

2022 春 ECNU

目标

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

实验要求

- 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。 例如：

```
#include <unistd.h>
……
chrt(10);/*  该程序将可以运行的最长时间为10秒，若没有运行结束，则强制结束*/
……
```

chrt的定义：

int chrt(long deadline);

/*deadline 是最后期限值(秒)，返回值1表示成功，返回值0表示该调用出错 */

实验要求(续)

- 在MINIX3中实现EDF近似实时调度功能：
  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。

实现说明

- 增加系统调用chrt：
- 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。

实现说明(续)

- 增加系统调用chrt：

1. 应用层：需要添加的系统调用chrt可以定义在unistd头文件中，并在libc中添加chrt函数体实现 。
   - 在/usr/src/include/unistd.h 中添加chrt函数定义。
   - 在/usr/src/minix/lib/libc/sys/chrt.c中添加chrt函数实现。可用alarm函数实现超时强制终止。参照该文件夹下fork.c文件，在实现中通过_syscall (调用号)向系统服务传递。例如：

```
pid_t fork(void)
{
    return(_syscall(PM_PROC_NR, PM_FORK, &m));
}
```

   - 在/usr/src/minix/lib/libc/sys中Makefile.inc文件添加chrt.c条目（添加C文件后，需在同目录下的Makefile/Makefile.inc中添加条目）。

实现说明(续)

- 增加系统调用chrt：

2. 服务层：需要向MINIX系统的进程管理服务中注册chrt，使得chrt服务可以向应用层提供 。
   - 在/usr/src/minix/servers/pm/proto.h中添加chrt函数定义。
   - 在/usr/src/minix/servers/pm/chrt.c中添加chrt函数实现，调用sys_chrt()
   - 在/usr/src/minix/include/minix/callnr.h中定义PM_CHRT编号。
   - 在/usr/src/minix/servers/pm/Makefile中添加chrt.c条目。

实现说明(续)

- 增加系统调用chrt：

2. 服务层：需要向MINIX系统的进程管理服务中注册chrt，使得chrt服务可以向应用层提供 。
   - 在/usr/src/minix/servers/pm/table.c 中调用映射表。
   - 在/usr/src/minix/include/minix/syslib.h 中添加sys_ chrt () 定义。
   - 在/usr/src/minix/lib/libsys/sys_chrt.c 中添加sys_chrt () 实现。可参照该文件夹下的sys_fork文件，在实现中通过_kernel_call (调用号)向内核传递。例如：

```
int sys_fork(parent, child, child_endpoint, flags, msgaddr)
{
  _kernel_call(SYS_FORK, &m);
}
```

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

实现说明(续)

- 增加系统调用chrt：

3. 内核层：在MINIX内核中实现进程调度功能，此处可以直接修改内核信息，例如进程的截至时间。
   - 在/usr/src/minix/kernel/system.h中添加do_chrt函数定义。
   - 在/usr/src/minix/kernel/system/do_ chrt.c中添加do_chrt函数实现。参考该文件下的do_fork文件，修改调用者进程信息。例如：

```
pid_t fork(void)
{
  return(_syscall(PM_PROC_NR, PM_FORK, &m));
}
```

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

实现说明(续)

- 增加系统调用chrt：

3. 内核层：在MINIX内核中实现进程调度功能，此处可以直接修改内核信息，例如进程的截至时间。
   - 在/usr/src/minix/include/minix/com.h中定义SYS_CHRT编号。
   - 在/usr/src/minix/kernel/system.c 中添加SYS_CHRT编号到do_chrt的映射。
   - 在/usr/src/minix/commands/service/parse.c的system_tab中添加名称编号对。

实现说明(续)

- MINIX3中的进程调度：
- MINIX3使用一种多级调度算法。进程优先级数字越小，优先级越高，根据优先级不同分成了16个可运行进程队列。每个队列内部采用时间片轮转调度，找到最高非空优先级队列，选取队列首部可运行的进程，当用完了时间片，则移到当前队列的队尾（详见教材P124）。
- 将EDF添加到多级调度算法中，可控制入队实现实时调度（也可有其他新颖方式，得分更高）。入队是将当前剩余时间（终止时间-运行时间）大于0的进程添加到某个优先级队列，即设置进程优先级（需要选择合适的优先级否则执行效果不理想）。
- 在该队列内部将时间片轮转调度改成剩余时间最少优先调度，即将剩余时间最小的进程移到队列首部。

实现说明(续)

- MINIX3中的进程调度：
- 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。
  - struct proc 维护每个进程的信息，用于调度决策。添加deadline成员。
  - switch_to_user() 选择进程进行切换。
  - enqueue_head() 按优先级将进程加入列队首。实验中需要将实时进程的优先级设置成合适的优先级。
  - enqueue() 按优先级将进程加入列队尾。同上。
  - pick_proc() 从队列中返回一个可调度的进程 。遍历设置的优先级队列，返回剩余时间最小并可运行的进程。

注意事项

1. MINIX的不同服务模块和内核都是运行在不同进程中，只能使用基于消息的进程间系统调用/内核调用，不能使用直接调用普通C函数。
2. 添加调用编号，需要修改取值范围限制。
3. 以源码为准（博客等资料版本落后）。
4. 善用source insight高级功能（调用关系，全局搜索）。
5. 善用git diff 检查代码修改。修改涉及文件较多，git diff可直观看到修改内容，避免引入无意的错误。
6. 善用FileZilla功能。连接虚拟机，拉取需修改的文件，修改后上传到虚拟机。

参考资料

1. 如何添加Minix内核调用
   https://wiki.minix3.org/doku.php?id=developersguide:newkernelcall
2. Minix内核调用APIs
   http://wiki.minix3.org/doku.php?id=developersguide:kernelapi
3. Minix讨论区
   https://groups.google.com/forum/#!forum/minix3

测试用例

- 在测试中，在main函数中fork三个子进程(P1, P2, P3)，并为每个子进程设置id。P1和P2为实时进程，deadline分别设为25s和15s。

  三个子进程会打印出子进程id和循环次数。

  第0s时：优先级 P2 > P1 > P3;

  第5s 时：P1设置deadline为5s，P1调用chrt(5);

  第5s后：优先级 P1 > P2 > P3;

  第10s时：P3设置deadline为3s，P3调用chrt(3);

  第10s后：优先级P3 > P2;
- 打印输出信息，观察子进程执行顺序是否正确。

```c
void proc(int id)
{
  int loop;
  switch (id)
  {
  case 1: //子进程1, 设置deadline=25
    chrt(25);
    printf("proc1 set success\n");
    sleep(1);
    break;
  case 2: //子进程2, 设置deadline=15
    chrt(15);
    printf("proc2 set success\n");
    sleep(1);
    break;
  case 3: //子进程3, 普通进程
    chrt(0);
    printf("proc3 set success\n");
    break;
  }
  for (loop = 1; loop < 40; loop++)
  {
    //子进程1在5s后设置deadline=5
    if (id == 1 && loop == 5)
    {
      chrt(5);
      printf("Change proc1 deadline to 5s\n");
    }
    //子进程3在10s后设置deadline=3
    if (id == 3 && loop == 10)
    {
      chrt(3);
      printf("Change proc3 deadline to 3s\n");
    }
    sleep(1); //睡眠, 否则会打印很多信息
    printf("prc%d heart beat %d\n", id, loop);
  }
  exit(0);
}
```

评分规则

- 完成项目功能的基础上，代码改动精简。
- 对原有功能/性能影响小。
- 代码风格接近原有MINIX。

实验提交

1. 提交材料：
   1. 该project的实验报告，测试代码。
   2. 相关源代码修改增量，内核补丁git diff > change.txt。
2. 实验要求：
   1. 独立完成实验，严禁抄袭。
   2. 请在规定时间内提交实验。