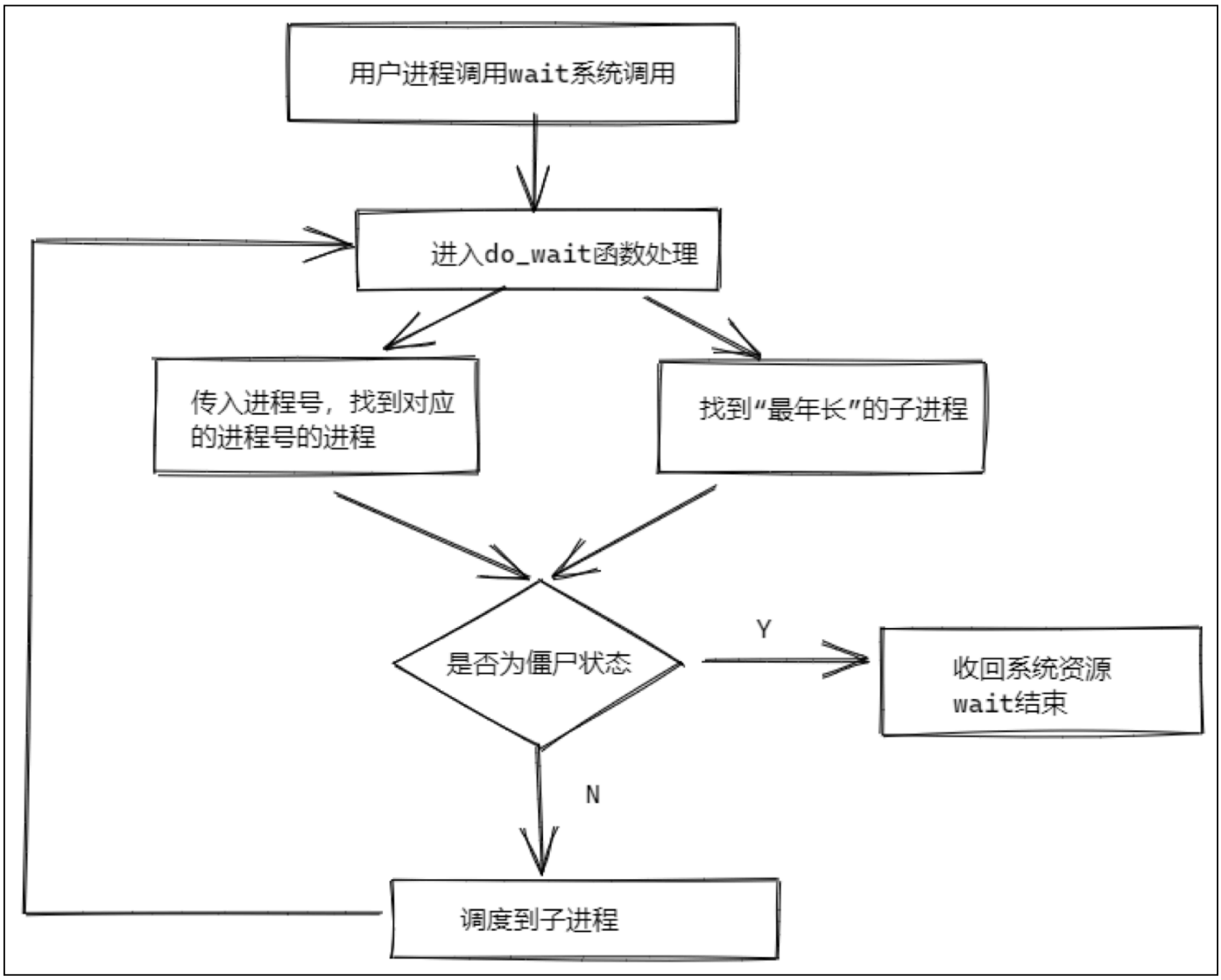
阅读week11代码，详细描述`user\rr.c`中产生的进程是如何进行调度的。描述中需要包含各进程的执行顺序，何时进入被调度的队列，何时被切换，执行结束时发生了什么



有一个父进程2以及子进程3、4、5、6、7

首先自然是父进程开始执行

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

#define TOTAL 5
/* to get enough accuracy, MAX_TIME (the running time of each process) should >1000
mseconds. */
#define MAX_TIME  10000
unsigned int acc[TOTAL];
int status[TOTAL];
```

```
int pids[TOTAL];

static void
spin_delay(void)
{
     int i;
     volatile int j;
     for (i = 0; i != 200; ++ i)
     {
          j = !j;
     }
}

int
main(void) {
     int i,time;
     memset(pids, 0, sizeof(pids));

     for (i = 0; i < TOTAL; i ++) {
          acc[i]=0;
          if ((pids[i] = fork()) == 0) {
               acc[i] = 0;
               while (1) {
                    spin_delay();
                    ++ acc[i];
                    if(acc[i]%4000==0) {
                        if((time=gettime_msec())>MAX_TIME) {
                            cprintf("child pid %d, acc %d, time
%d\n",getpid(),acc[i],time);
                            exit(acc[i]);
                        }
                    }
               }
          }
          if (pids[i] < 0) {
               goto failed;
          }
     }

     cprintf("main: fork ok,now need to wait pids.\n");

     for (i = 0; i < TOTAL; i ++) {
         status[i]=0;
         waitpid(pids[i],&status[i]);
         //cprintf("main: pid %d, acc %d, time %d\n",pids[i],status[i],gettime_msec());
     }
     cprintf("main: wait pids over\n");
     return 0;
```

```
failed:
     for (i = 0; i < TOTAL; i ++) {
          if (pids[i] > 0) {
               kill(pids[i]);
          }
     }
     panic("FAIL: T.T\n");
}
```

之后在fork命令之下，3-7号被加入队列，因为都是无限循环的进程，他们会一直运行（本该），但由于设定了最长运行时间，在运行到一定时间会被强制退出。

每次时钟中断剩余的时间片就会-1， 如果为零，则会设置为需要调度的状态，在trap.c中的trap函数中会进行调度。这时候就调度到4号进

程。重复这个过程。3，4，5，6，7，3，4，5，6，7，……（RR调度算法）。直到某一个进程执行结 束，由于我们创建的子进程是相同的，这里是3号进程最先执行结束。

父进程会在waitpid等子进程，当子进程寄（僵尸）了他会等下一个子进程直到全寄