Implementation of virtual time in kernel

Variable added in ***sched_entity*** structure :

volatile u64 on_cpu_time;

On_cpu_time stores the CPU execution time for self task.

volatile u64 mx_on_cpu_time;

mx_on_cpu_time stores the maximum CPU execution time over all child task including itself.

u64 del_exec;
u64 base_on_cpu_time;
u64 naive_vtime;
u64 mx_naive_vtime;

These variable were used for testing purpose.

struct list_head child_vtime_at_exit;
spinlock_t child_vtlist_lock;

Every time a child task terminates it reports its on_cpu_time to its parent by enqueuing own on_cpu_time into parent's child_vtime_at_exit list. The lock is used to ensure mutual exclusion while multiple children try to access the list concurrently.

Structure ***vtime_struct*** added in ***/include/linux/list.h file*** to store the virtual time in the list :

```
struct vtime_struct {
                        pid_t pid;
                        u64 vtime ;
                        struct list_head next;
                    } ;
```

The added variables in ***sched_entity*** structure are initialized in ***__sched_fork()*** function:

```
p->se.on_cpu_time               = current->se.on_cpu_time;
p->se.base_on_cpu_time          = current->se.on_cpu_time;
p->se.mx_on_cpu_time            = 0;
INIT_LIST_HEAD(&p->se.child_vtime_at_exit);
p->se.naive_vtime               = 0;
p->se.mx_naive_vtime            = 0;
spin_lock_init(&p->se.child_vtlist_lock);
```

Updating the ***on_cpu_time*** variable in ***update_curr()*** function:

```
curr->on_cpu_time += delta_exec;
curr->del_exec = delta_exec;
curr->naive_vtime += delta_exec;
```

Updating the ***mx_on_cpu_time*** variable in ***do_exit()*** function:

```
struct task_struct *parent = tsk->group_leader;
    if (parent->se.mx_on_cpu_time <= tsk->se.on_cpu_time){
        parent->se.mx_on_cpu_time = tsk->se.on_cpu_time;
    }
    parent->se.mx_naive_vtime = parent->se.mx_naive_vtime > tsk->se.naive_vtime?
parent->se.mx_naive_vtime : tsk->se.naive_vtime;
```

Enqueuing own ***on_cpu_time*** in the parent's ***child_vtime_at_exit*** list at the task termination in ***do_exit()*** function:

```
extern int pthread_join_activated;
```

```
        if (pthread_join_activated >0 ){
                struct vtime_struct *tmp;
                struct list_head *pos, *q;
                // delete every child vtime record
                list_for_each_safe(pos, q, &current->se.child_vtime_at_exit){
                        tmp= list_entry(pos, struct vtime_struct, next);
                        //printk(KERN_INFO "freeing item pid= %d\n", tmp->pid);
                        list_del(pos);
                        kfree(tmp);
                }



                if (tsk->pid != parent->pid){

                        struct vtime_struct *vtst = kmalloc(sizeof(struct
vtime_struct),GFP_KERNEL);
                        if (vtst){
                                vtst->pid = tsk->pid;
                                vtst->vtime = tsk->se.on_cpu_time;
                                //kfree(vtst);
                                spin_lock(&parent->se.child_vtlist_lock);
                                list_add ( &vtst->next , &parent->se.child_vtime_at_exit ) ;
                                spin_unlock(&parent->se.child_vtlist_lock);

                        }

                }
        }
```

System call added to the kernel (/kernel/sys.c) to communicate from userspace:

```
SYSCALL_DEFINE1(group_vtime, int, arg0)//335 //return max v_time in
thread group
{
	struct pid *pid_struct;
	struct task_struct *t;
	pid_struct = find_get_pid(arg0);
	t = pid_task(pid_struct,PIDTYPE_PID);
	struct task_struct *parent = t->group_leader;
	u64 mx_time = parent->se.mx_on_cpu_time;
	struct task_struct *temp = parent;
	do{
		mx_time = mx_time > temp->se.on_cpu_time?
mx_time:temp->se.on_cpu_time;
		temp = next_thread(temp);
	}while(temp != parent);

	return mx_time-parent->se.base_on_cpu_time;
/*

	if(parent->se.mx_on_cpu_time > parent->se.on_cpu_time){
//	printk(KERN_INFO "group_vtime syscall called with %d \treturn:
%llu\n",arg0,parent->se.mx_on_cpu_time  );

		return parent->se.mx_on_cpu_time -
parent->se.base_on_cpu_time;
	}else{
//		printk(KERN_INFO "group_vtime syscall called with %d
\treturn: %llu\n",arg0,parent->se.on_cpu_time  );
```

```
                return parent->se.on_cpu_time -
parent->se.base_on_cpu_time;
        }
*/


}


SYSCALL_DEFINE2(add_vtime, int, arg0, long long int, delta_v)//334
{
        struct pid *pid_struct;
        struct task_struct *t;
        pid_struct = find_get_pid(arg0);
        t = pid_task(pid_struct,PIDTYPE_PID);
         t->se.on_cpu_time +=delta_v;

//  printk(KERN_INFO "add_vtime syscall called with %d \tdelta:
%d\n",arg0,delta_v  );
 return t->se.on_cpu_time ;
}



SYSCALL_DEFINE2(set_vtime, int, arg0, long long int, v_time)//338
{
        struct pid *pid_struct;
        struct task_struct *t;
        pid_struct = find_get_pid(arg0);
        t = pid_task(pid_struct,PIDTYPE_PID);
         t->se.on_cpu_time = v_time;
```

```
//  printk(KERN_INFO "add_vtime syscall called with %d \tdelta:
%d\n",arg0,delta_v  );
 return t->se.on_cpu_time ;
}


SYSCALL_DEFINE1(sync_vt_at_join, int, child_pid)//340
{




/*  struct vtime_struct vtst = {
               .pid = (pid_t)child_pid,
               .vtime = 10,
               .next = LIST_HEAD_INIT(vtst.next)
       } ;
*/
       extern int pthread_join_activated;
       if(child_pid == -1){//activate pthread_join
              pthread_join_activated = 1;
              return current->se.on_cpu_time ;

       }
       if (pthread_join_activated > 0){
              struct vtime_struct *tmp;
              struct list_head *pos, *q;

              list_for_each_safe(pos, q,
&current->se.child_vtime_at_exit){
              tmp= list_entry(pos, struct vtime_struct, next);
```

```
                    printk(KERN_INFO "freeing item pid= %d
child_vtime:%llu parent_vtime:%llu\n",
tmp->pid,tmp->vtime,current->se.on_cpu_time);
                    current->se.on_cpu_time =
current->se.on_cpu_time >
tmp->vtime?current->se.on_cpu_time:tmp->vtime;
                    spin_lock(&current->se.child_vtlist_lock);
                    list_del(pos);
                    spin_unlock(&current->se.child_vtlist_lock);
                    kfree(tmp);
                }
        }
//  printk(KERN_INFO "add_vtime syscall called with %d \tdelta:
%d\n",arg0,delta_v  );
 return current->se.on_cpu_time ;
}



SYSCALL_DEFINE1(del_exec, int, arg0)//336
{
        struct pid *pid_struct;
        struct task_struct *t;
        pid_struct = find_get_pid(arg0);
        t = pid_task(pid_struct,PIDTYPE_PID);

//  printk(KERN_INFO "add_vtime syscall called with %d \tdelta:
%d\n",arg0,delta_v  );
 return t->se.del_exec ;
}

SYSCALL_DEFINE1(exec_start_gtime, int, arg0)//337
```

```
{
        struct pid *pid_struct;
        struct task_struct *t;
        pid_struct = find_get_pid(arg0);
        t = pid_task(pid_struct,PIDTYPE_PID);

//  printk(KERN_INFO "add_vtime syscall called with %d \tdelta:
%d\n",arg0,delta_v  );
 return t->se.exec_start_gtime ;
}



SYSCALL_DEFINE3(copy_times,
                unsigned long , num_thread,
                unsigned long *, thread_times,
                int, arg0) //339
{

        unsigned long vtimes [num_thread+1][3]; //(unsigned long*)
kmalloc(sizeof(unsigned long),GFP_KERNEL);

        struct pid *pid_struct;
        struct task_struct *initial_task,*t;

        pid_struct = find_get_pid(arg0);
        initial_task = pid_task(pid_struct,PIDTYPE_PID);

        t = initial_task;
        int i = 1;
```

```
        do{
                vtimes[i][0] = t->pid;
                vtimes[i][1] = t->se.on_cpu_time;
                vtimes[i][2] = t->se.exec_start_gtime;

                t = next_thread(t);
                i++;
          }while(t!=initial_task);

        vtimes[0][0] = i-1;
   if (copy_to_user(thread_times, &vtimes,
(num_thread+1)*3*sizeof(unsigned long)));
      // return -EFAULT;

   /* return amount of data copied */
   return i-1;
}

SYSCALL_DEFINE1(naive_vtime, int, arg0) //341
{
        struct pid *pid_struct;
        struct task_struct *t;
        pid_struct = find_get_pid(arg0);
        if(pid_struct == NULL) {return 1;}
        //t =find_task_by_vpid(arg0);//
        t =  pid_task(pid_struct,PIDTYPE_PID);
        if(t == NULL) {return 1;}
```

```
//  printk(KERN_INFO "v_time syscall called with %d \t @ v_time
%llu\n",arg0,t->se.on_cpu_time  );
 return t->se.mx_naive_vtime > t->se.naive_vtime ?
t->se.mx_naive_vtime: t->se.naive_vtime ;




}




SYSCALL_DEFINE2(sync_max_vtime,
                int, num_thread,
                int*, thread_pid,
                ) //342
{
        int i;
        struct pid *pid_struct;
        struct task_struct *task;
        u64 mx = 0;
        for (i = 0; i< num_thread; i++){
                pid_struct = find_get_pid(thread_pid[i]);
        task = pid_task(pid_struct,PIDTYPE_PID);
                mx = task->se.on_cpu_time > mx ? task->se.on_cpu_time :
mx;

        }
        for (i = 0; i< num_thread; i++){
                pid_struct = find_get_pid(thread_pid[i]);
        task = pid_task(pid_struct,PIDTYPE_PID);
                task->se.on_cpu_time =  mx;
```

```
        }


    }
```