Forking a New Process with the Predetermined PID Value during Process Migration

Narayan A. Joshi
Assistant Professor
Computer Science Department
Institute of Science & Technology for Advanced Studies & Research
Vallabh Vidyanagar

D. B. Choksi
Professor & Head
Post Graduate Department of Computer Science & Applications
Sardar Patel University
Vallabh Vidyanagar

ABSTRACT
In dynamic process migration, a process which is under execution is migrated from its originating machine to a new machine in order to uphold load-balancing in large distributed systems by allowing applications to access a large number of computing resources spread across the network and may provide significant improvements in overall system behavior. After migration of a process to the destination workstation, the resuming process must be allocated the same 'process-id value': the value which the process originally possessed on the source workstation before migration. This paper discusses a kernel-level solution for the problem of checkpointing the credential of 'process identifier' and to allocate a selected process-id value to a new process. We suggest a new system call setforkpid( ) to assign the specific process-id value of interest to the selected new process. An attempt is made to suggest a solution for the Linux kernel 2.6.25 environment.

General Terms
Load Balancing, Operating Systems

Keywords
Process migration, Process id, Process credentials, Process checkpointing and restart, setforkpid system call.

1. INTRODUCTION
A binary program image is a description of a computation which is executable by a computer. A process is a dynamic entity of this binary program-image that is actually under execution in the underlying operating system platform. During the process execution, the process-state consists of two interconnected components: a static state and a dynamic state. The binary program image i.e. the executable program file characterizes the static state of the process. Whereas the various dynamic features like: the process credentials, the hardware i.e. CPU-registers (both for process and operating system), the stack (both for process and operating system), process virtual memory, process physical memory, occupied and requested I/O devices, system call under execution, regular-files opened, pipes-opened, signals received, blocked and pending, sockets established, IPCs, currently working threads and such threads' state and other such terms characterize the dynamic process state [1],[4].

In specific circumstances, the system administrator, the system user, a third party application software or the underlying operating system itself require to recognize or capture the process's dynamic state information. Next section we discusses about some of such circumstances.

Section 2 briefly describes the scenarios which motivated us to conduct the study. Section 3 draws attention to the scope of problem and its significance. Section 4 focuses on the theoretical mechanism and our solution for Linux operating system. Section 5 presents conclusion and directions for future work.

2. MOTIVATIONS
In certain circumstances it becomes advantageous to identify the Process's dynamic state information. Here some of the circumstances are discussed:

Process control and debugging: Process examination, or the ability to recognize a process-state and modify its execution, is a fundamental requirement for system utilities such as debuggers and security tools. Certain system programs do require capabilities to collect the credentials of a desired process. In certain circumstances such programs would like to collect the values of the process credentials like process-identifier (pid), user-identifier (uid), group-identifier (gid), effective user-identifier (euid) and suid, fsuid, sgid, fsgid and many others which are currently allocated to or possessed by the process. As well the programs may prefer that the later restarting process must possess the same credential values which it possessed during its initial start.

Process checkpoint-restart to perform process migration: The optimization techniques such as load balancing and process migration are used in a network of workstations to relocate the load from tightly loaded workstations towards the lightly loaded workstations. Such procedures are also appropriate where the preferred resource by a process is not distantly accessible or the remote utilization of the resource produces reduced performance.

In such circumstances the system performance can be improved by relocating the executing process itself to the host machine where the desired resource is accessible. Also the processes having large execution period of time may require to be migrated to provide reliability and fault tolerance, in order to face a certain class of faults about which some notification in advance can be achieved. For example, either the system administrator or the operating system may in advance send notification to process that the system is about to shut down. In all these situations, where the dynamic process migration is desirable, the process first needs to be checkpointed and later to be restarted on different machine. The process checkpointing mechanism needs to pull together the recent dynamic state information of the process [2]-[4], [10]-[12].

3. THE PROBLEM
The dynamic state information of a process is maintained by an operating system in a kernel-level data structure called Process Control Block (PCB). The PCB maintains process-

state details like process-credentials, current state, address-space, list of virtual memory regions, currently opened files, signals, signal handlers and much more [1], [2].

In contrast to the static migration of the process, the dynamic migration of the process is superior to the static migration of the process. One of the important characteristics of dynamic migration of a process indicates that the process must resume and continue its execution on the destination workstation with the same identity that it possessed when it was checkpointed on the source workstation.

At the process startup time, the process is assigned a system vide unique process id by the host operating system. The unique process id is the only facility available in user-space through which the associated process can be made enabled to be controllable or traceable by either the owner of the process or the system administrator. As well, the application programmers may manage the process by the associated unique process id through the application program also.

Therefore, the checkpointing mechanism must consider the checkpointing of the unique process id possessed by the process being executed. The mechanism must support not only checkpointing the unique process id but also make arrangements such that on resumption on the destination workstation, the process must continue execution with the same unique process id.

Although the PCB maintains the process identifier value owned by the specific process, the Linux kernel 2.6.25 does not avail a kernel-level facility through which the newly created process can be started with a specific allocated value of the process identifier. This paper describes a mechanism to assign a specific value to the process identifier of a desired new process. This paper presents a mechanism to checkpoint the unique process id credential possessed by a process under execution, to resume the process with the same credential of process id on destination workstation.

We present the mechanism for the Linux operating system with kernel version 2.6.25 for the i686 SMP platform. In this paper, we have discussed a new mechanism to inject the process id which is specified by us in to the kernel space (during process resumption on the destination workstation) by introducing our own new system call in the operating system's kernel; so that the resuming process on the destination workstation will start with the process id which it possessed initially before process migration. The proposed mechanism requires modification of kernel source code and addition of a new system call; and recompilation of the kernel.

4. MECHANISM

Before we present the steps necessary for establishing a particular process id value to the process descriptor, we outline here the working of a process id.

UNIX processes are always assigned a number to uniquely identify them in their namespace. This number is called the process identification number or PID for short. Each process generated with fork or clone is automatically assigned a new unique PID value by the kernel; which in later can not be tampered, and such a tampering may cause harm to the process. In a thread-less process, the PID and TGID of a process are identical. The main process in a thread group is called the group leader. The group_leader element of the task structures of all cloned threads points to the task_struct instance of the group leader. [1].

The global PID and TGID are directly stored in the UNIX's PCB task_struct, namely, in the elements pid and tgid:

```
<sched.h>
struct task_struct {
    ...
    pid_t pid;
    pid_t tgid;
    ...

}
```

Both are of type pid_t, which resolves to the type __kernel_pid_t; this, in turn, has to be defined by each architecture. Usually an unsigned integer is used, which means that 232 different IDs can be used simultaneously.[1], [5]

The checkpointing of the existing value of the process id can be performed with the help of :

```
struct tast_struct *p=find_task_by_pid(pid);
task_lock(p);
        /* Now checkpoint values of credentials of the
desired process e.g.
        p->pid,
        p->uid and others..
        */

task_unlock(p);
```

To restore a process on destination workstation with desired process id; we present here a mechanism to introduce our own system call setforkpid() whose invocation before forking a new process will set a specific process id of our interest to the process descriptor of the newly created process:

1. Enter the following line in the kernel's unistd_32.h file.

   #define __NR_setforkpid 327

   Here we create a constant and assign a unique system-call number by which our system call is known to the Linux kernel.
2. Enter the following system call function prototype in the syscalls.h file.

   asmlinkage long sys_setforkpid (pid_t forkpid);

   Here we create a prototype for our system-call function; the system-call implementation and the user-space invocation to our system call setforkpid() must match the specified prototype.

3. Enter the following line in the syscall_table_32.S file at the end of the file.

   .long sys_setforkpid

   This represents a unique array-index position in the system call table which is responsible to hold the address of the our specified system call's implementation function at system startup and later on.

Create a file setforkpid.c in the new directory syscall_setforkpid in the kernel source tree and enter the following code to the file.

```
#include<linux/linkage.h>
#include<linux/types.h>
#include<linux/unistd.h>
#include<linux/module.h>
#include<linux/kernel.h>

extern void set_specific_pid(pid_t forkpid);

asmlinkage long sys_setforkpid(pid_t forkpid)
{
    …
    //printk(KERN_DEBUG
"sys_setforkpid():parameter forkpid=%d.\n",
forkpid);

    set_specific_pid(forkpid);
    …

    return forkpid;
}
```

Where, the extern function set_specific_pid and some other extern functions have been exported by us in the kernel source; as shown below:

4. Update the Makefile. Append the path of the above created directory syscall_setforkpid to the kernel's compilation path.

5. Also add the following lines to the kernel's source which are responsible to maintain the specific process id that is passed to the kernel-space from the user-space:

```
pid_t specific_pid = 0;

pid_t get_specific_pid();
EXPORT_SYMBOL(get_specific_pid);
pid_t get_specific_pid()
{
    return specific_pid;
}

void reset_specific_pid();
EXPORT_SYMBOL(reset_specific_pid);
void reset_specific_pid()
{
    specific_pid=0;
}

void set_specific_pid(pid_t the_pid);
EXPORT_SYMBOL(set_specific_pid);
void set_specific_pid(pid_t the_pid)
{
    specific_pid = the_pid;
    …
    //set pid here
    //set tgid here
    …
}
```

Where, the kernel symbol specific_pid of type pid_t is responsible to hold the value arriving to kernel-space from the user-space.

7. Recompile this updated kernel source and reboot with the newly compiled kernel.

[6]-[9]. Here are the steps to be followed by the user-space process on the destination workstation to establish a new process with a desired process id value [11],[12]:

1. Unpack the received checkpoint image on the destination workstation.
2. Obtain the checkpointed value of the process id which is to be assigned to the newly created process, say it as specific_pid.
3. Execute our own system call as following: **setforkpid(specific_pid);**

5. CONCLUSION

An attempt is made here to checkpoint the process credential; and restore and establish the same value of credential at the process restart on the destination workstation via the mechanism of provisioning of a new system call. We believe that the method represented here will be beneficial not only to the checkpointing applications and system debuggers but also to the system administrators and the system programmers.

6. REFERENCES

[1] Daniel P. Bovet and Marco Cesati: "Understanding the Linux Kernel", 3rd editon, O'Reilly publication, 2005.
[2] D. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou: "Process migration"; ACM Computing Surveys; 32(3):241-299, 2000
[3] D. Nichols: "Using idle workstation in a shared computing environment"; Proceedings of the eleventh ACM symposium on operating system principles; ACM; November – 1988; pp 512
[4] F. Douglis and J. Ousterhout: "Process migration in the sprite operating system"; Proceedings of the 7th international conference on distributed computing systems, IEEE, Berlin, West Germany, September 1987, pp. 1825
[5] Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman: "LINUX Device Drivers", 3rd edition, O'Reilly publication.
[6] Linux Kernel source
[7] Linux Reference Manual, Section 2, ptrace.
[8] Linux Reference Manual, Section 5, proc.
[9] Linux Source File: /usr/include/asm/unistd.h in Linux source code.
[10] M. Kozuch and M. Satyanarayanan: "Internet suspend/resume"; Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, IEEE CS Press, 2002, pp. 40–46
[11] N. A. Joshi and D. B. Choksi; "Process Forensic For System-call Details on Linux Platform"; International

Journal Of Computer Applications in Engineering, Technology & Sciences; November-2009; pp-510-512

[12] N. A. Joshi and D. B. Choksi; "Checkpointing Process Virtual Memory Area on Linux Platform"; International journal of Emerging Technologies and Applications in Engineering Technology and Sciences; June-2010; pp-42-44