CHECKPOINT AND RESTART MECHANISM

(CSE 316 – OPERATING SYSTEMS)

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

SR. NO.	NAME	ROLL No.	REG. No.
1	MOHAMMED AMJAD	K18 PD B62	11803910

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

ACKNOWLEDGEMENT

I would like to thank my mentor - Ms. Amandeep Kaur for her advice and inputs on this project. Many thanks to my friends and seniors as well, who spent countless hours to listen and provide feedbacks. This project let me study more deep about the operating system.

INTRODUCTION

A process checkpoint/restart mechanism allows a user to stop a process, save its state to disk, copy that state to another machine, and restart that process the new machine. Here we building a linux driver system which is checkpoint and restart mechanism .

in checkpoint The state usually includes register set, address space, allocated resources, and other related process private data. Restart work as it resumes the process execution by restoring the checkpointed state of the process, on the destination machine.

For making this mechanism we used handles like atcheckpoint() and atrestart()

Also we want to use some libraries for it function control like

ckpt_create(),ckpt_remove,ckpt_stat.

Name: Mohammed Amjad

Reg. No: 11803910

Email: amjadkooriyad@gmail.com

GitHub Link : https://github.com/AJ10LAB/checkpoint-.git

Algorithm: Checkpoint and Restart mechanism

Q:30

Description:

Checkpoint and Restart mechanism

Checkpointing a given process is nothing but saving its state. The state usually includes register set, address space, allocated resources, and other related process private data. Restart mechanism resumes the process execution by restoring the checkpointed state of the process, on the destination machine

Ideally the checkpointing system should be able to save and restore the following:
- State of memory and CPU registers.
- Child processes of a checkpointed process.
- The status of open file descriptors held by the checkpointed process

Use of checkpoint mechanism

- Improve a system's load balancing and scheduling
- Run complex simulation or modelling applications

Approaches to Checkpointing:

1.user level
2. kernel level
3.application level

User level Checkpointing

Application programs to be checkpointed are linked with a checkpoint user library. Upon checkpointing, a checkpoint-triggering signal is sent to the process. The functions in the checkpoint library responds to the signal and save the information necessary to restart the process. On restart, the functions in the checkpoint library restore the execution environment for the process.

Kernel-level Checkpointing
In kernel-level checkpointing, the operating system supports checkpointing and restarting processes. The checkpointing is transparent to applications; they do not have to be modified or linked with any special library to support checkpointing

Application-level Checkpointing
Applications can be coded in a way to checkpoint themselves either periodically or in response to signals sent by other processes. When restarted, the application must look for the checkpoint files and restore its state.

Problem Statement

To design and implement a transparent checkpoint /restart Mechanism for Linux Operating System

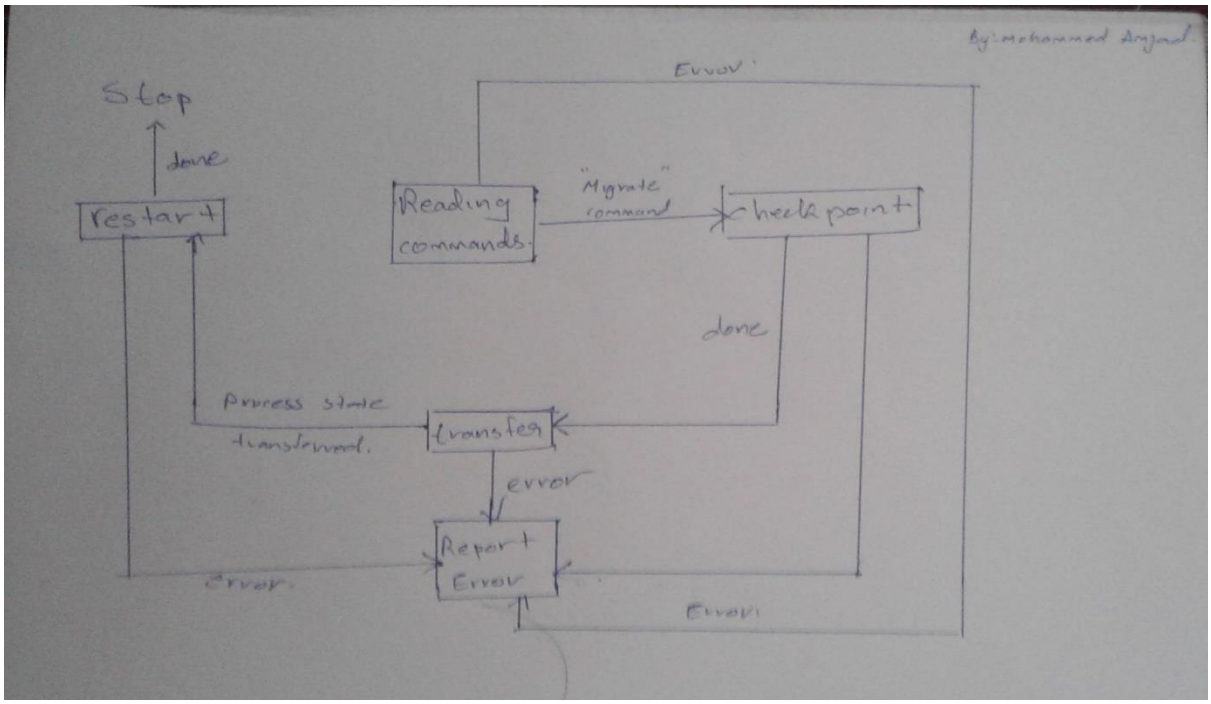
Problem Description

Process migration is the relocation of a process from its current location to another node to migrate a process, we need to checkpoint its state, transfer the state to the destination node and restart the process at the node. The basic part is the checkpoint /restart system, which deals only with the mechanism of checkpointing a process, and restarting the one checkpointed earlier. It does of course provide a clean interface, but it is the duty of the upper layers to define their own policies. A wide range of policies might be needed, depending on whether the main concern is load sharing, load balancing or application concurrency.

Structured Analysis

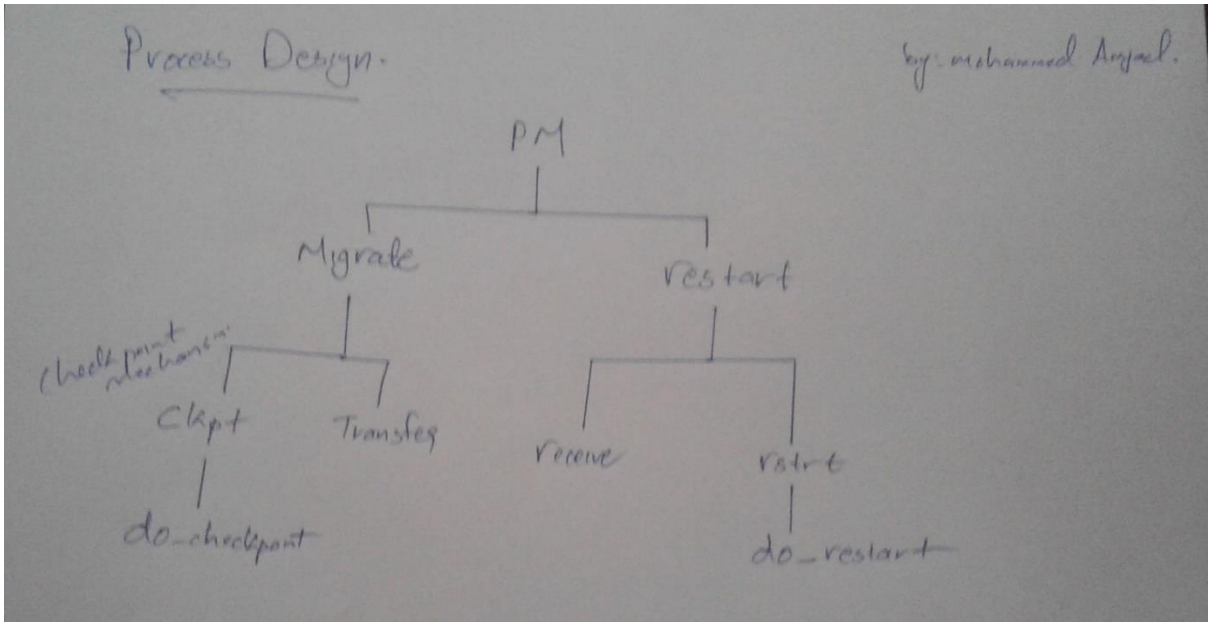
The process migration system is intended to migrate the given process to the given destination node. The major sub activities involved in process migration are
1. Checkpointing (suspending and saving the state) the migrating process on the source node.
2. Transferring the process state to the destination node.
3. Restarting (resuming the execution) the process on the destination node based on the saved state, from exactly the point of suspension.

Diagram:



The process starts when the command is given at the terminal. When the checkpoint signal is given the process is checkpointed and the image file is created. The image file is transferred to another machine and it is restarted. At any time during the course of this process, if any errors arise, appropriate error messages are displayed to the user.

Process Design:



Checkpoint

1. select the process to be checkpointed
2. if the process is not a checkpointable process then go to step 6
3. freeze the execution of the process
4. collect and save all the data regarding current state of the process
5. resume or stop the execution of the freeze process depending on the option given
6. stop

restart

1. if the file containing the checkpointed image is not a valid checkpoint file, go to step 6
2. create a new process
3. change the working directory of the created process to the one that is saved in the checkpointed image
4. read the state of the checkpointed process from the file and restore it in the new process
5. change the state of the process as “runnable”
6. stop

Implementation

As already said the checkpointing can be done at three levels, and here it is done in kernel level. To checkpoint any arbitrary process, the checkpointing process must be in the kernel mode. And for this it has to make a system call. But as of now the Linux OS do not provide system calls for checkpointing a process. Adding a system call to the OS is not that simple and also requires change of source code, which violates our primary design goal. But most operating systems support dynamically loadable kernel modules. Kernel modules are typically loaded at boot time or on demand when the kernel requires certain functionality. They provide some entry points similar to system calls with which we can enter into kernel mode. Once the module is loaded, it becomes part of the kernel address space and can access everything in the kernel

.

Code:

[*] checkpoint mechanism.c

```
1  #include <stdlib.h>
2  #include <ckpt.h>
3  #include <sys/mount.h>
4  #include <sys/types.h>
5  #include <unistd.h>
6  #include <stdio.h>
7
8  extern void ckptSocket(void);
9  extern void ckptXserver(void);
10 extern void restartSocket(void);
11 extern void restartXserver(void);
12 extern char *cdFile;
13 extern FILE fpCD;
14 extern char *cdFile;
15 extern FILE fpCD;
16 extern long cdOffset;
17 long cdOffset;
18 extern int sock;
19
20 catchRESTART()
21 {
22     while (access("/CDROM/data", R_OK) == -1) {
23         perror("Insert CDROM");
24         sleep(60);
25     }
26     if ((fCD= fopen&#40;cdFile, "r"&#41;) == NULL)
27         perror("can't open CDFile"), exit(1);
28     if (fseek(fCD, cdOffset, SEEK_SET))
29         perror("can't seek to CDOffset"), exit(1);
30 }
31 catchCKPT()
```

rces | Compile Log | Debug | Find Results

Sel: 0 | Lines: 113 | Length: 2853 | Insert | Done parsing in 0.031 seconds

[*] checkpoint mechanism.c

```
31  catchCKPT()
32  {
33      alarm(60);
34      close(sock);
35      alarm(0);
36      cdOffset = ftell(fCD);
37      fclose(fCD);
38      umount("/CDROM");
39      exit(0);
40  }
41
42  static int
43
44  do_checkpoint(ckpt_id_t id, u_long type, char *path_name)
45  {
46      int cr;
47      printf("CKP%d, type %s,to dir%s n",
48          id, ckpt_type_str(CKPT_REAL_TYPE(type)), path_name);
49      if ((cr = ckpt_create(path_name, id, type, 0, 0)) != 0) {
50          printf("Failed to checkpointing process %lldn", id);
51          return (cr);
52      }
53      return (0);
54  }
55
56  do_restart(char *path_n)
57  {
58      printf("Restarting remaining processes %s n", path_n);
59      if (ckpt_restart(path_n, 0, 0) < 0) {
60          printf("Restarting %s failedn", path_n);
61          return (-1);
```

Compile Log | Debug | Find Results

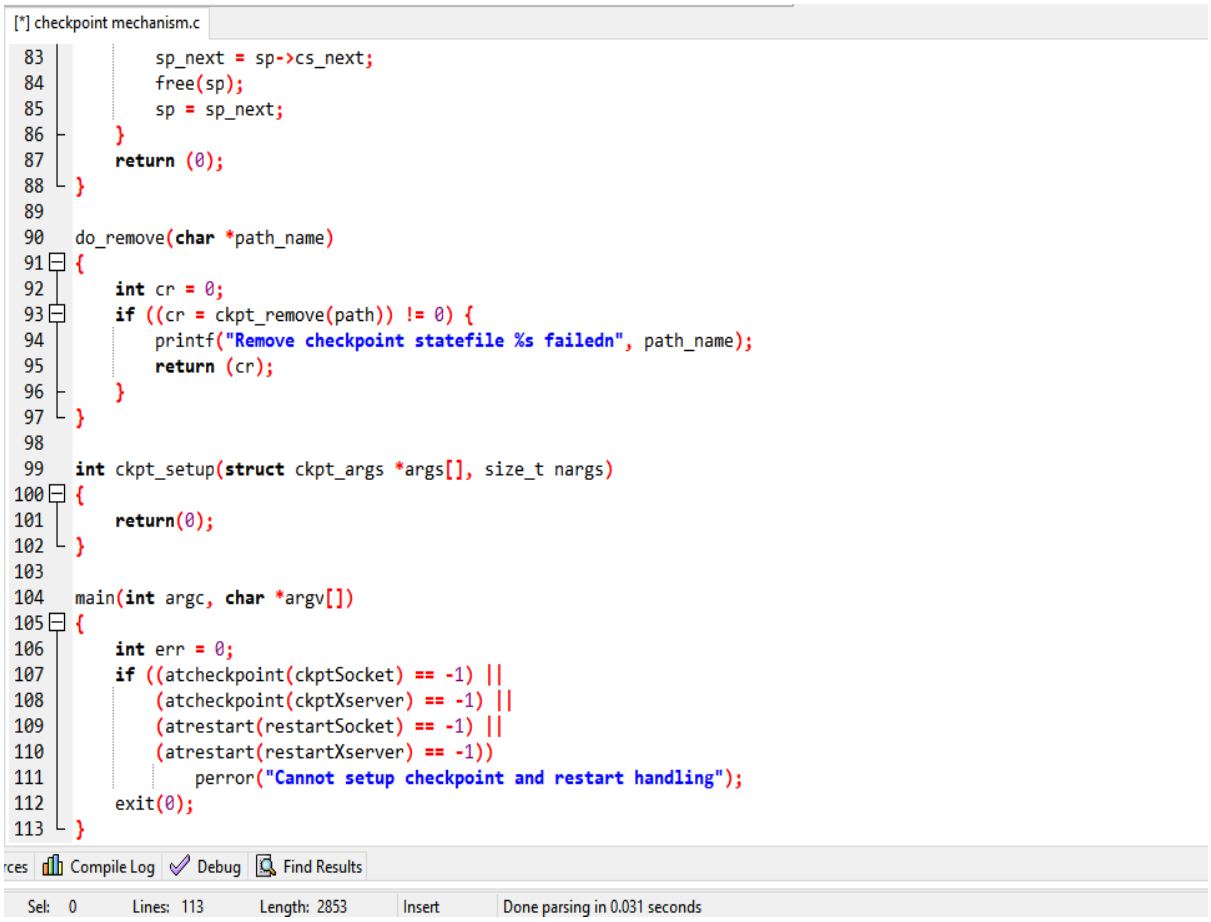
Sel: 0 Lines: 113 Length: 2853 Insert Done parsing in 0.031 seconds

[*] checkpoint mechanism.c

```
58      printf("Restarting remaining processes %s n", path_n);
59      if (ckpt_restart(path_n, 0, 0) < 0) {
60          printf("Restarting %s failedn", path_n);
61          return (-1);
62      }
63  }
64
65  ckpt_info(char *path)
66  {
67      ckpt_stat_t *sp, *sp_next;
68      int cr;
69      if ((cr = ckpt_stat(path, &sp)) != 0)
70      {
71          printf("Not able to get info about CPR file %s \n", path);
72          return (cr);
73      }
74      printf("nInformation About Statefile %s (%s):\n",
75          path, rev_to_str(sp->cs_revision));
76      while (sp) {
77          printf(" Process:tt%sn", sp->cs_psargs);
78          printf(" PID,PPID:tt%d,%dn", sp->cs_pid, sp->cs_ppid);
79          printf(" PGRP,SID:tt%d,%dn", sp->cs_pgrp, sp->cs_sid);
80          printf(" Working at dir:t%sn", sp->cs_cdir);
81          printf(" Num of Openfiles:t%dn", sp->cs_nfiles);
82          printf(" Checkpointed @t%sn", ctime(&sp->cs_stat.st_mtime));
83          sp_next = sp->cs_next;
84          free(sp);
85          sp = sp_next;
86      }
87      return (0);
88  }
```

Compile Log | Debug | Find Results

Sel: 0 Lines: 113 Length: 2853 Insert Done parsing in 0.031 seconds



Explanation – line by line explanation:

(104-113)

it calls atcheckpoint() and atrestart() to set up functions for handling checkpoint and restart events.

The checkpoint and restart library interfaces are contained in the libcpr.so so that we want to include a library #include<ckpt.h>

When we use atcheckpoint() and atrestart() we ensure that the signal handles are invoke only when checkpoint and and restart mechanism on running state

It is like preparing for a checkpoint mechanism.

Time Complexity: 5 it mean O(1)

(31-40):

Next we want to do is to handle this checkpoint ,so we making catchCKPT() .it marks the current file position in cdFile as open ,then it save for restore at restart time then closes cdFile and unmount the cd rom
Then in same we want to add a alarm in callback. The alarm system call send signal to calling program after some second that we specified. We set for 60 second otherwise it get exited

Time Complexity: 1 it mean O(1)

(20-30):
Next we want to add restart handler to it. We making a code that wait for cd-rom to become mounted, then it reopen the cdfile and move or seek the offset positin in cdfile

Time Complexity: 4 it mean O(1)

(44-54):
Here we using ckpt_create() for saving state, function of ckpt_create() is it checkpoint a process or set of processes into a state file. For that we maken do_chechpoint() function.
Time Complexity: 2 it mean O(1)

(56-63):
After saving state next we want to resume the mechanism, so that we using ckpt_restart() it function is it resumes execution of checkpointed process or processes.
For that we make a do_restart() function then we add ckpt_restart in it.
Time Complexity: 2 it mean O(1)

(65-88):
Next we want to check the status or the current condition of the mechanism. Here we using ckpt_stat() for checking status. It function is it retrieves status information about a checkpoint state file.
For that we making a ckpt_info() function and adding ckpt_stat() to it

Time Complexity: 3 it mean O(1)

(90-97):

If any failure occur or we want to retrieve from checkpoint mechanism that when we feel it get failed to remove a certain checkpoint we using ckpt_remove. It delets a checkpoint state file directory.

Time Complexity: 2 it mean O(1)

(99-102):

If we want to know the current implementation of the ckpt_create() function we want to add extra handler that is ckpt_setup().

Time Complexity: 1 it mean O(1)

CONCLUSION

In this project we maken code for checkpoint and restart mechanism. Which work in different levels of mechanism. First it reading the command then it going to checkpoint if it wantto restart it go to restart mode if there is an error it show error. The all mechanism is a time bounded we added particulat time to it to work the mechanism . a process checkpoint/restart mechanism allows a user to stop a process, save its state to disk, copy that state to another machine, and restart that process the new machine. Build a Linux device driver that implements a process checkpoint/restart mechanism this is the main function of it.

PLAGIARISM CHECKER: