# README

**This file provides the documentation and the listing of each file and a little explanation of each and also the steps to test the code**.

My goal for the semester was to write a “preprocessor” that takes a “c” file reads it and for each function defined in that file, if finds out the variables that this function depends upon. This functionality is very important for the main project that we were working on, which is to figure out the impact on the performance on the Invite system, if it knows whether it has seen the current state or not and if it has, then it skips the test. My code comes into play, when we talk about a state. We decided to differentiate among the states, based on the values for each variable, the current test depends upon. My code finds out these variables and after the integration, will pass along these variables to the next “preprocessor that creates the resulting code for testing.

I divided the code in multiple files with each file housing the code called on the similar levels. I

### The List of Files and Directories:

### The CKPT directory

**session.py** 🡪 goes in the : **/dejaview/utils/manage/djvw\_manage/utils/** deirectory.

It is important to replace existing file with this one, for this version makes it possible for the child to know its ID. Since the original manager (dejaview) doesn’t offer to run any scripts while resuming, there is no sure fire way to know the child id other than this.

**mig.o utils.o zaplib.o zutils.o 🡪** needed for use of ZAP structures and functions in ckpt( )

**ckpt.h & ckpt.c 🡪** The header file declares the function **ckpt( )** and the C code file defines it. The .c file also contains another local function that returns the child id. They are well documented and describe what they are doing. The documentation for the ckpt( ) and get\_childID( ) can also be seen in the Section 5 where I talk about both of them.

**testckpt.c 🡪** this is the test file that calls the ckpt( ) function and if the return value is zero, it writes ‘I am the Parent’ to the Parent\_Output file, and loops forever. However, if the return value is the positive number, it copies the return value (child ID) in a variable and writes ‘I am the Child # childID” and loops on it forever.

**treeckpt.c 🡪** this test creates a tree like output, where the main function initializes two variables top and bottom to zero and then calls ckpt( ) and if the return value is > 0 it sets top = child ID. And then it goes on to ckpt( ) again and this time for return value > 0 it sets bottom = child ID. So after branch and resume we get different values in the output file for each.

**Makefile** 🡪 makes all the targets to the copied to the appropriate pods.

### The CKPT\_Invite Directory

**testinviteckpt.c testinviteckpt.ic ckpt.h and { ckpt.c (slightly different form the original) }**

These are the same files that existed in the original invite code as demo.c etc. Just the name is different. (The changes were made to precompile.pl, for **InVite**’s integration with the **ckpt( )** )

**./precompile.pl testinviteckpt.c testinviteckpt.ic**

Gives us ckpt\_outpt.c that is the final C source file that has the **ckpt( )** functionality built in.

**Makefile** 🡪 makes the final binary that can be run inside the pod to test the functionality.

### The Test Scripts:

**start**  🡪 starts of three pods/sessions and adds the treeckpt, testckpt and ckpt\_output one in each. (Now we have three pods that have already check pointed)

**restart\_test new\_session\_name** 🡪 branches off of the testckpt session and resumes the child

**Parent\_Output\_test** 🡪 prints out the output of the parent process for testckpt.

**Child\_Output\_test pod\_number** 🡪 prints out the output of the child inside the pod specified (Works for testckpt)

**restart\_tree** 🡪 branches off of both the checkpoints taken by treeckpt and resumes them, then it branches off of the first child and resumes that child of the child.

**Tree\_Output pod\_number pod\_number pod\_number** 🡪 prints the output produced by the entire tree, given the pod numbers for the children.

**restart\_invite parent\_session\_name parent\_pod\_number** 🡪 branches off of the ckpt\_output into three child sessions and resumes them

**Invite\_Parent\_Output** 🡪 prints the output of the parent process only

**Invite\_Output pod\_number pod\_number pod\_number** 🡪 give it the pod numbers for the child processes and it prints the output for the parent as well as the children.

**delete\_session session\_name** 🡪 stop then delete any session

**delete\_tree** 🡪 delete the tree created by the restart\_tree script

**delete\_invite** 🡪 delete the child sessions created by the restart\_invite

**delete\_tests** 🡪 delete the sessions created by the start script

**addtest addtree ./CKPT\_Invite/addinvite** are three helper scripts that the user doesn’t need to call directly.

**\* Important note: Make Sure the permissions for all the scripts are set to 755 (chmod 755 script\_name and put the CKPT directory in home directory, for example : /home/CKPT.**

**\*\*One Last Thing: ZAP is limited to the number of pods per system runs/mounts. Meaning, if you restart too many checkpoints, you will have to reboot the system for the pods to start smoothly.**

Remember that changing the order of execution of these scripts between output and delete may not always work.