	Max	Actual	Comments
Functionality (points earned/lost based on running your program)	**50**	**0**	
Where/What/When backup policies are appropriate, justified, and properly	6		Maxvers check and succesful mount and umount
Backup files' creation is as efficient as possible.	8		For N backups getting created
Visibility policy: backup versions of files are not accessible by default. Can not be easily viewed, manipulated, or deleted	8		On ls, the files should not shown in stdout. (4 pts) Should not allow to be opened by vim. (2 pts) Should not allow rm to delete the version file. (2 pts)
Retention policy is reasonable and properly enforced	8		For Oldest backup to be removed on exceeding Nth Backup
Version management functions (5pts max for list, del, view, and restore)	20		All Functionalities 5 pts Each 1) List option for the versions available 2) Delete a particular version from the versions available 3) View a particular version and able to see its contents 4) Restore a previous content to be latest and then able to view it through opening by vim.
Code, Compilation, Mounting, Module	**25**	**0**	
Code compiles without any warnings	4		No warnings (-1 per warning)
Your code is written in good kernel style with comments.	5		No Comments - 0
File system mounts/unmounts smoothly with required options and checks for incorrect	2		mount and umount option (1 each)
User code supports all arguments, checks for invalid argument combinations, and	4		4 validation for bkpctl
Test scripts that exercise each feature of your bkpfs. Scripts should have ample	10		10 different test scripts - each 1 mark
Reliability and Effectiveness	**10**	**0**	
No (possible) deadlocks/races noticed, or other issues affecting system stability.	5		For Every other Error -2
No memory/reference leaks noticed	5		For Every Slab Error -2
Documentation and Submission	**15**	**0**	
README (design doc) is clean and readable. Describes the design and reason it. No	15		Design Decisions - (When , Where , How Backups are created)
Extra Credit	**35**	**0**	
Space-based retention policy	10		
Capture meta-data file changes	10		
wrapfs bug fixes (optional)	10		
Grader's discretion for clever solutions, enhancements, test scripts, or other extras.	5		
General Demerits (use negative numbers)	**0**	**0**	
Followed GIT submission guidelines improperly.	0		
Submission on time: deduct 1 point for every late hour (time rounded up in units of	0		
Kernel does not crash. Each (different) kernel crash costs 3 point	0		
Total Grade (out of 100)	**100**	**0**	
Total Extra Credit (NOT counted as part of the total above)	**35**	**0**	

1) Move a file into the mounted path
2) We edit it for N consecutive backups.
3) We test it for user program ioctls.
4) Create 2 more backups to see if oldest version is removed
5) Umount and Mount again with maxvers commandline param.
6) Check again for backups created

Note :
write_data.sh which helps to write number of instances and bytes to write for each data
Check if the files are created by having a watch over the /test/higherpath and /test/lowerpath