CS4444 Project 2

Shawn Cross, Ryan Crane

October 30, 2017

We plan to modified the the dispatch and add functions from the original noop scheduler to implement the LOOK algorithm for the scheduler. We made it so that you can go both forward and backward in the dispatch function when finding the next request to dispatch. If the distance to a request in the opposite direction is closer than a request in the current direction the direction will change. The queue sorted after a request has been dispatched. If the queue is empty then we add the new requests directly to the queue. If there is already requests in the queue then when adding then we sort during the insertion. the requests are inserted into the queue based on position.

1 Version Control log

Detail	Author	Description
88e93dc	Ryan Crane	dining philosophers problem, looks correct
35c460c	Shawn Cross	adding the write up for the concurrency problem.
3f0111c	Shawn Cross	More done on the write up and made it so you can compile for the makefile.
2d47960	Shawn Cross	slight change to tex file.
9aa562c	narxy	Update philosopher.tex
f8b1617	narxy	Update philosopher.tex
651df07	Shawn Cross	fixed spelling errors.
7555688	Shawn Cross	uploading tarball and created the sstf-iosched.c file
4bc64ff	Shawn Cross	Uptdate to the Kconfig file, update to the Makefile, update to menuconfig, and sstf-iosched.

2 Work Log

Thursday 10/26 6:00 pm to 8:00 pm

started assignment mostly reading up on Kernel.

Friday 10/27 12:00 pm to 4:00 pm

modified config and makefiles to adopt our new scheduler.

Sunday 10/29 12:00 pm to 5:00 pm

Implemented Look algorithm and started the LaTeXdocument.

Monday 10/9 12:00 pm to 2:00 pm

Finished the LaTeXdocument.

3 Main point of the assignment

We believe that the main point of the assignment was to understand how requests are dealt with in an operating system. It was also begin to understand how to interact with the kernel.

4 How we approached the problem

First we started by figuring out how to get our scheduler working with the kernel instead of the noop scheduler. Once we were able to do that we began to figure out how we were going to implement the look schedluer. We knew we needed to be able move forward and backward and that there needed to be two sorts. First we firgured out how to get the forward and backward working by checking if the the next request in the direction that we were currently going is closer than the next request in the oppsite direction. If it was closer than we continue in that direction and if it was futher away then we change direction and go to the closer request. After a request is dispatched then we will re-sort the queue. We also needed to modify the add function so that we would sort the requests as they were inserted into the queue. We did this by using an insertion sort as requests were added into the queue.

5 Correctness and Testing

We know that our secheduler was being used because we used the command
cat /sys/block/hda/queue/scheduler
and got an output
noop [sstf] deadline cfq
this shows that our sstf scheduler is being used as we wanted. We then added print statements to our scheduler code so that we could determine when requests were being added and dispatched from the queue and by doing this we were able to show that for every requeust that was being added to the queue it was also being dispatched as well which means that all requests were being dealt with. We also added print statements to declare when the direction flag had its value changed to show that we were in fact changing direction. With all of these things working as we expected them to we were able to prove that our look scheduler was properly working the way we wanted it to.

6 What we learned

We learned that you have to make modification to multiple files when adding new features to the operating system. We also learned that writing code for the kernel can be hard and confusing because we could not find a lot of information on some of the built-in function being used so it took us some time trying to figure what they did and how they worked. We also learned that can be a ton of I/O requests and they can all happen very quickly. We also learned that there are many different ways to do the scheduling.

7 TA Evaluation

Steps for applying patch to clean linux-yocto-3.19 tree.

1. step 1: clone new repo from github using the command

   git clone git://git.yoctoproject.org/linux-yocto-3.19

2. step 2: switch to the 3.19.2 by using the command

   git checkout -b linux-yocto-3.19.2

3. step 3: copy the file /scratch/fall2017/files/config-3.19.2-yocto-qemu into a file named .config into the source root of the repo that you just cloned.

4. step 4: Now source the environment configuration. If you are using bash you will need to source /scratch/fall2017/files/environment-setup-i586-poky-linux and if you are using tcsh you will need to source /scratch/fall2017/files/environment-setup-i586-poky-linux.csh

5. step 5: You need to apply the patch we provided to the repo by using the command

   git apply [path to patch file] where path to patch file is the path you need to the patch file that we have provided.

6. step 6: You now need to make the kernel by running the command make -j4

7. step 7: Once the kernel is created you will need to run the vm using the command

   qemu-system-i386 -gdb tcp::???? -S -nographic -kernel [path to bzImage] -drive file=[path to core image] -enable-kvm -net none -usb -localtime –no-reboot –append "root=/dev/hda rw console=ttyS0 debug

   Where [path to core image] is the file found in

   /scratch/fall2017/files/core-image-lsb-sdk-qemux86.ext4

   and the [path to bzImage] for me would be the bzImage found in

   linux-yocto-3.19/arch/x86/boot/bzImage

   and the ???? are the port number that you want to connect gdb to remotely.

   After this command is run the terminal will halt.

8. step 8: you will now need to run $GDB in a new terminal window. You may need soucre the enviroment again as you did in step 4 previously if you did start a new termial session and the $GDB command does not work. Once gdb is running you will use the command

   target remote :????

   where the ???? are the same as the port that you had specified in step 7.

9. step 9: Once you have gdb is remotely connected you can type continue and the terminal with the vm halted will begin to boot. You will be asked if you want to use the sstf scheduler. You should be able to press y and then enter and and you will be promted about which scheduler to use again. I beleive our is option 4 but pick the number that is next to the sstf option. The booting will continue after this and eventually you will see our print statements about requests being added and dispatched. You should evtually see a login promt but you may also miss it from all of the print statements. Wait a few seconds and then eventually the print statements will stop. type root to login and you will begin to see more print statement. You will notice that some will have first request added and then immidiatley see a statemet about the request being dispatched. You will then see points at which multiple requests are added and then see that our scheduler is in fact moving forward and backward. and eventually the statements will stop coming and you can then look and see that our schedular is working as we had wanted it to by examining the previous statements that were outputted to the terminal.

10. step 10: To shutdown the vm you simply need to type reboot and press enter. Once the vm has shut down you can then exit gdb by typing quit into the terminal that you have gdb open in.