

CSE-344 SYSTEM PROGRAMMING
SPRING 2024
HOMEWORK 5
REPORT

ONUR ATASEVER
210104004087

This homework and report is modified according to barriers. In the worker threads, int the beginnin barrier is used

I applied worker manager approach for threads as requested. I created the Manager, the desired number of threads and the desired size buffer. The manager calls the copy_directory function and adds it to the buffer. Since the place where the buffer is added is a region that can be accessed by all threads, it is considered a critical region and is protected with a mutex. The buffer size was checked with each addition to the buffer. If the buffer is full, the thread goes to sleep and continues to block until it receives the buffer is not full signal. When the buffer is not full, every time an element is added, the buffer is not empty signal is sent. Thus, if there is a worker waiting by blocking it because the buffer is empty, it is woken up. If what is wanted to be written to the buffer is a folder rather than a file, this folder is recursively sent back to the copy_directory function.

First thread is at the beginning of the Worker threads. So all the other threads waits the other threads arrive that point. Worker threads check the buffer size when they start running. If the buffer is empty, they are blocked and start waiting. After receiving the Buffer is not empty signal, it wakes up and continues to do its job. Retrieves data from the buffer and deletes it. According to the information they receive from Buffer, they determine the file type and increase the necessary counters. Mutex protections are used in every operation regarding the buffer as it is a critical area and to provide synchronization. Then, copying is done from the source path to the destination path. It is copying byte by byte. So with valgrind it takes time to run. Second barrier is in the end of the worker function. So all the workers wait in the end before they return.

Signal handler created for CTRL+C. Each thread runs when the running variable is 1. If this signal comes, the running variable becomes 0 and the threads stop running. In addition, blocked threads are woken up in the signal handler so that the program reaches the end properly, performs the freeing operations and ends.

To run test 1 with valgrind you can use “make valgrind1” command, for test 2 “make valgrind2” and for test 3 “make valgrind3”.

For each read, write, open system calls for file, error check is done. For opening directory or getting file status error checks are done. For sigaction function, error checks is done and fort he command line arguments error check is done.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
==32148== Memcheck, a memory error detector
==32148== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==32148== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==32148== Command: ./MwCp
==32148==
Usage: ./MwCp <buffer_size> <num_workers> <src_dir> <dest_dir>
==32148==
==32148== HEAP SUMMARY:
==32148==     in use at exit: 0 bytes in 0 blocks
==32148==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==32148==
==32148== All heap blocks were freed -- no leaks are possible
==32148==
==32148== For lists of detected and suppressed errors, rerun with: -s
==32148== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
make: *** [Makefile:30: runV3] Error 1
root@DESKTOP-PLJPMRB:/home/hw4/code#
```

Pseudo Code For Main:

```
FUNCTION main(argc, argv)
    IF argc != 5 THEN
        PRINT "Usage: %s <buffer_size> <num_workers> <src_dir> <dest_dir>"
        EXIT_FAILURE

    SETUP signal handler for SIGINT to call inHandler

    PARSE command line arguments to get buffer_size, num_workers, src_dir,
dest_dir

    ALLOCATE memory for worker threads

    INITIALIZE buffer with buffer_size

    GET current time as start time

    CREATE manager thread to run manager

    function FOR each worker thread
        CREATE worker thread to run worker function

    JOIN manager thread

    FOR each worker thread
        JOIN worker thread

    GET current time as end time

    CALCULATE execution time

    PRINT statistics: number of files, directories, total bytes copied, execution time

    FREE allocated memory for worker threads

    DESTROY buffer

    RETURN 0
END FUNCTION
```

Pseudo Code For Manager:

```
FUNCTION manager(arg)
    CALL copy_directory(src_dir, dest_dir)
    LOCK buffer.mutex
    SET buffer.done to 1
    BROADCAST buffer.not_empty // Wake up all worker threads
    UNLOCK buffer.mutex
    RETURN NULL
END FUNCTION
```

Pseudo Code For copy_directory:

```
FUNCTION copy_directory(src, dest)
    OPEN source directory
    CREATE destination directory
    WHILE there are more entries in the source directory AND running is
        TRUE IF entry is not "." or ".." THEN
            CONSTRUCT src_path and dest_path
            GET file status
            IF file is a regular file THEN
                ADD file pair to buffer
            ELSE IF file is a directory THEN
                INCREMENT directory count
                RECURSIVELY call copy_directory(src_path, dest_path)
        CLOSE source directory
    END FUNCTION
```

Pseudo Code For Worker:

FUNCTION worker(arg)

 WHILE running IS TRUE

 REMOVE file pair from buffer

 IF file pair is empty THEN

 BREAK

 OPEN source and destination files

 COPY file contents from source to destination

 CLOSE source and destination files

 RETURN NULL

END FUNCTION

Mutex Protection And Conditional wait Examples:

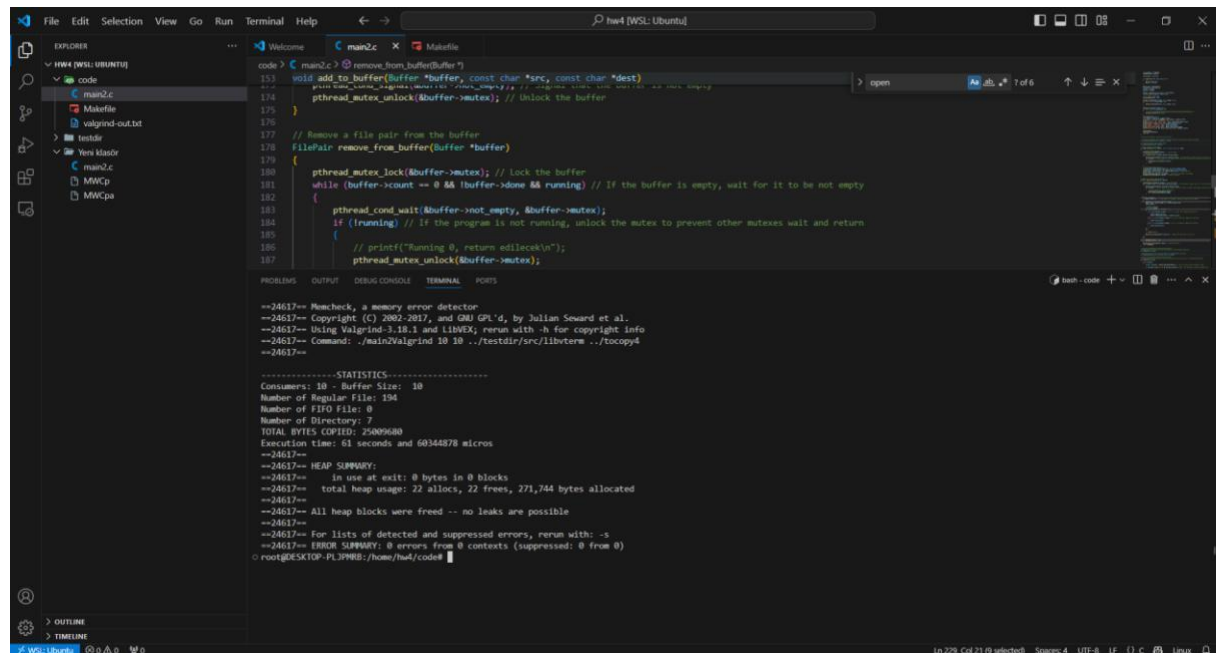
```
// Add a file pair to the buffer
void add_to_buffer(Buffer *buffer, const char *src, const char *dest)
{
    // printf("Locktan önce\n");
    pthread_mutex_lock(&buffer->mutex); // Lock the buffer
    // printf("Locktan sonra\n");
    while (buffer->count == buffer->buffer_size && running)
    {
        pthread_cond_wait(&buffer->not_full, &buffer->mutex); // If the buffer is full, wait for it to be not full
        if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
        {
            pthread_mutex_unlock(&buffer->mutex);
            return;
        }
    }
    // printf("While'dan sonra\n");
    strncpy(buffer->buffer[buffer->tail].src, src, MAX_FILENAME_LENGTH);
    strncpy(buffer->buffer[buffer->tail].dest, dest, MAX_FILENAME_LENGTH);
    buffer->tail = (buffer->tail + 1) % buffer->buffer_size; // Update the tail
    buffer->count++; // Increment the count
    pthread_cond_signal(&buffer->not_empty); // Signal that the buffer is not empty
    pthread_mutex_unlock(&buffer->mutex); // Unlock the buffer
}
```

```
// Remove a file pair from the buffer
FilePair remove_from_buffer(Buffer *buffer)
{
    pthread_mutex_lock(&buffer->mutex); // Lock the buffer
    while (buffer->count == 0 && !buffer->done && running) // If the buffer is empty, wait for it to be not empty
    {
        pthread_cond_wait(&buffer->not_empty, &buffer->mutex);
        if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
        {
            // printf("Running 0, return edilecek\n");
            pthread_mutex_unlock(&buffer->mutex);
            return;
        }
    }
}
```

Signal Handler Example For CTRL+C:

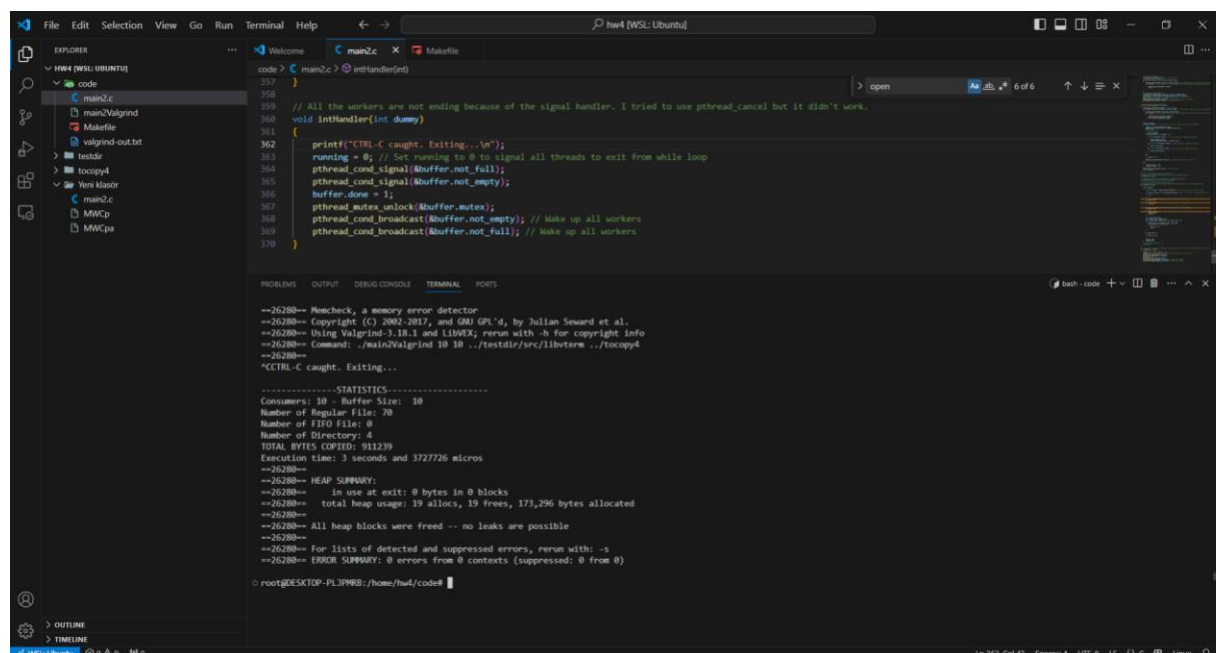
```
// All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
void inHandler(int dummy)
{
    running = 0; // Set running to 0 to signal all threads to exit from while loop
    pthread_cond_signal(&buffer.not_full);
    pthread_cond_signal(&buffer.not_empty);
    buffer.done = 1;
    pthread_mutex_unlock(&buffer.mutex);
    pthread_cond_broadcast(&buffer.not_empty); // Wake up all workers
    pthread_cond_broadcast(&buffer.not_full); // Wake up all workers
}
```

Test 1 Output:



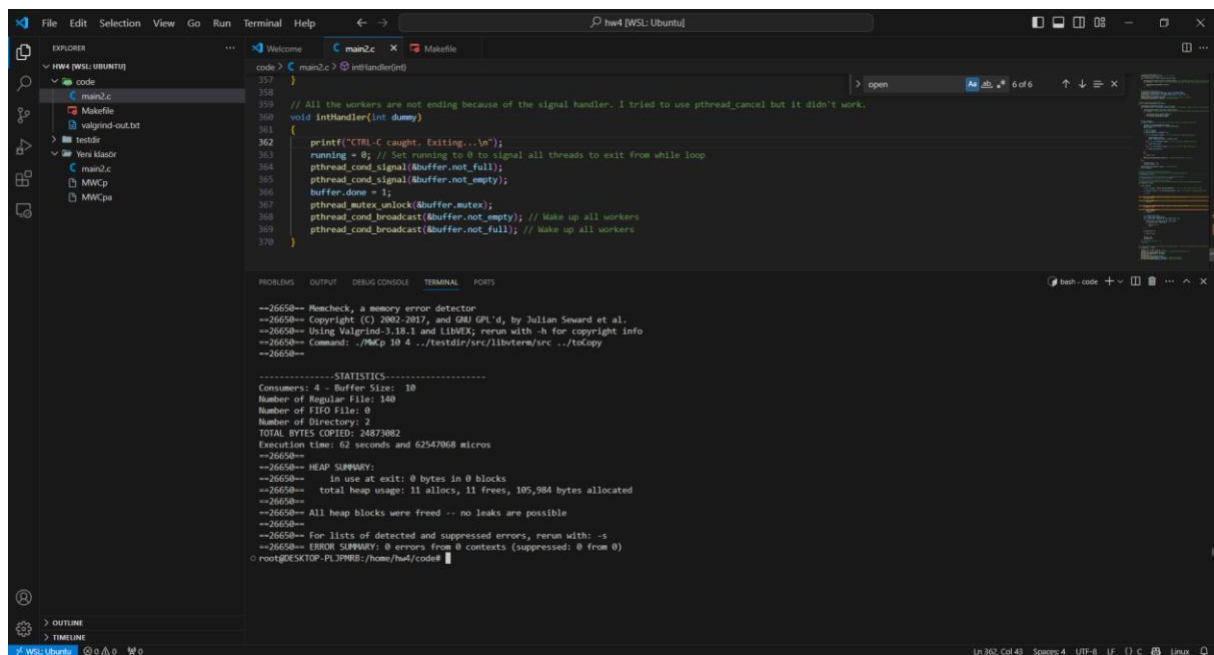
```
code > C:\main2.c > remove_from_buffer(buffer)
153 void add_to_buffer(buffer *buffer, const char *src, const char *dest)
154 {
155     pthread_mutex_lock(&buffer->mutex); // Lock the buffer
156     while (buffer->count == 0 && !buffer->done && running) // If the buffer is empty, wait for it to be not empty
157     {
158         pthread_cond_wait(&buffer->not_empty, &buffer->mutex);
159     }
160     if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
161     {
162         printf("Running 0, return edilock\n");
163         pthread_mutex_unlock(&buffer->mutex);
164     }
165 }
166
167 // Remove a file pair from the buffer
168 FilePair remove_from_buffer(buffer *buffer)
169 {
170     pthread_mutex_lock(&buffer->mutex); // Lock the buffer
171     while (buffer->count == 0 && !buffer->done && running) // If the buffer is empty, wait for it to be not empty
172     {
173         pthread_cond_wait(&buffer->not_empty, &buffer->mutex);
174     }
175     if (!running) // If the program is not running, unlock the mutex to prevent other mutexes wait and return
176     {
177         printf("Running 0, return edilock\n");
178         pthread_mutex_unlock(&buffer->mutex);
179     }
180 }
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Test 1 Output With CTRL+C:



```
code > C:\main2.c > inHandler(int)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void inHandler(int dummy)
361 {
362     printf("CTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer.not_full);
365     pthread_cond_signal(&buffer.not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer.not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer.not_full); // Wake up all workers
370 }
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Test 2 Output:

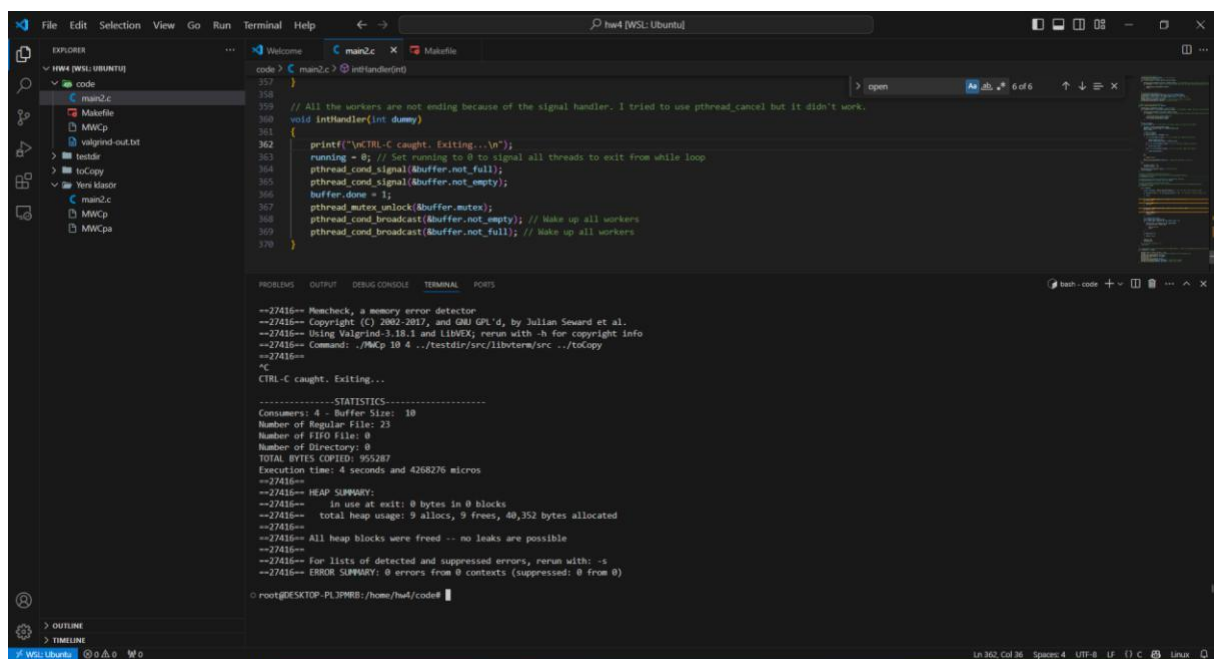


```
code > C:man2.c > intHandler(int)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("CTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer_not_full);
365     pthread_cond_signal(&buffer_not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer_not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer_not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--26658-- Memcheck, a memory error detector
--26658-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--26658-- Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
--26658-- Command: ./MKp 10 4 ../testdir/src/libterm/src ../toCopy
--26658--

-----STATISTICS-----
Consumers: 4 - Buffer Size: 10
Number of Regular File: 140
Number of FIFO File: 0
Number of Directory: 2
TOTAL BYTES COPIED: 24873082
Execution time: 62 seconds and 62547068 micros
--26658--
--26658-- HEAP SUMMARY:
--26658--   in use at exit: 0 bytes in 0 blocks
--26658--   total heap usage: 11 allocs, 11 frees, 105,984 bytes allocated
--26658--
--26658-- All heap blocks were freed -- no leaks are possible
--26658--
--26658-- For lists of detected and suppressed errors, rerun with: -s
--26658-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PL3PMBR: /home/hw4/code#
```

Test 2 Output With CTRL+C:

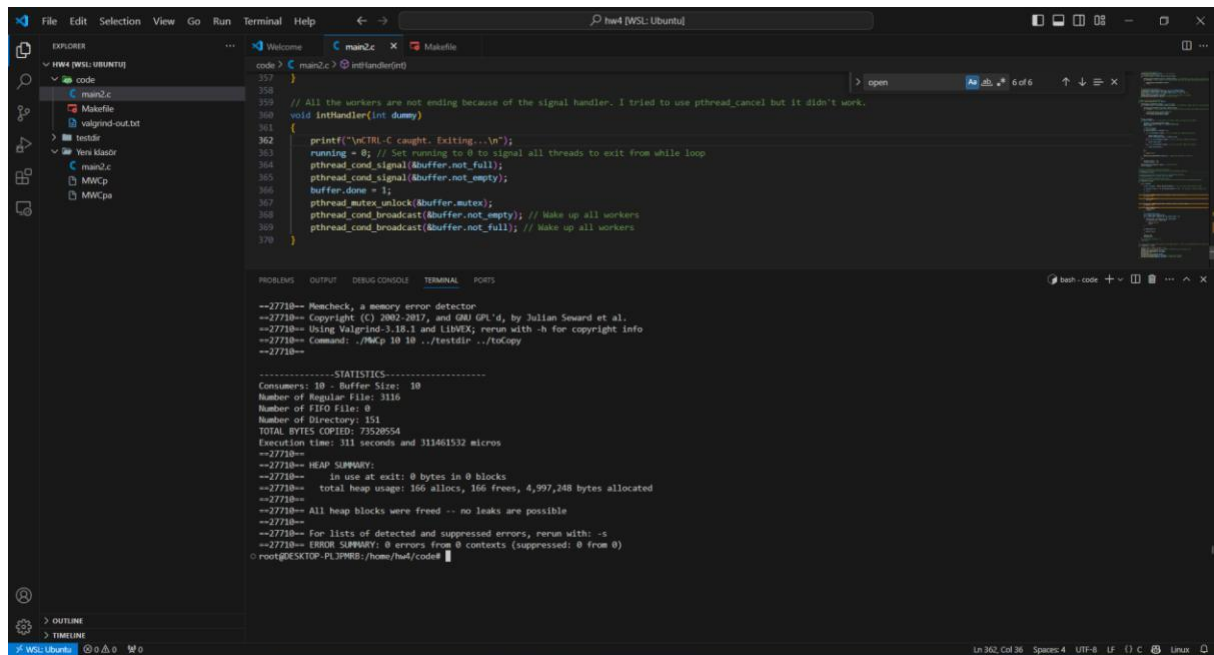


```
code > C:man2.c > intHandler(int)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("\nCTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer_not_full);
365     pthread_cond_signal(&buffer_not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer_not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer_not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--27416-- Memcheck, a memory error detector
--27416-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--27416-- Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
--27416-- Command: ./MKp 10 4 ../testdir/src/libterm/src ../toCopy
--27416--
CTRL-C caught. Exiting...

-----STATISTICS-----
Consumers: 4 - Buffer Size: 10
Number of Regular File: 23
Number of FIFO File: 0
Number of Directory: 0
TOTAL BYTES COPIED: 955207
Execution time: 4 seconds and 4268276 micros
--27416--
--27416-- HEAP SUMMARY:
--27416--   in use at exit: 0 bytes in 0 blocks
--27416--   total heap usage: 9 allocs, 9 frees, 40,352 bytes allocated
--27416--
--27416-- All heap blocks were freed -- no leaks are possible
--27416--
--27416-- For lists of detected and suppressed errors, rerun with: -s
--27416-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PL3PMBR: /home/hw4/code#
```

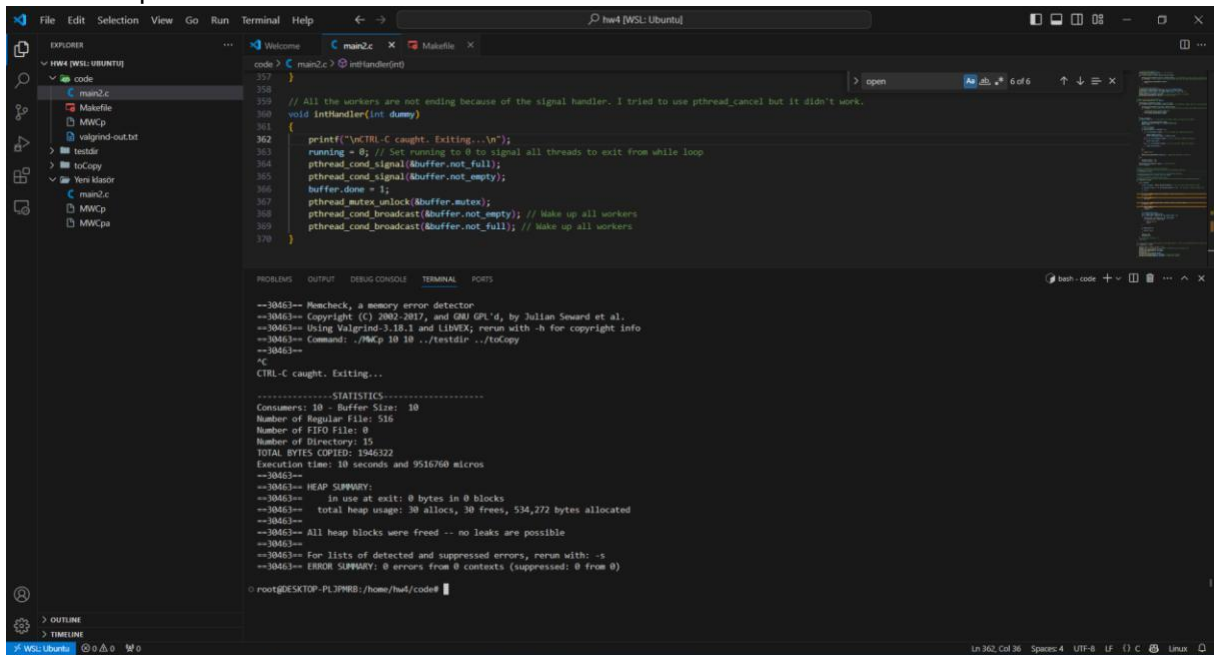

Test 3 Output:



```
code > C:\main2.c > intHandler(int)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("\nCTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer_not_full);
365     pthread_cond_signal(&buffer_not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer_not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer_not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--27718-- Ponchek, a memory error detector
--27718-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--27718-- Using Valgrind-3.18.1 and LDBEX; rerun with -h for copyright info
--27718-- Command: ./MKP 10 10 ../testdir ../toCopy
--27718--
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular File: 3116
Number of FIFO File: 0
Number of Directory: 151
TOTAL BYTES COPIED: 7352054
Execution time: 311 seconds and 311461532 micros
--27718--
--27718-- HEAP SUMMARY:
--27718--   in use at exit: 0 bytes in 0 blocks
--27718--   total heap usage: 166 allocs, 166 frees, 4,997,248 bytes allocated
--27718--
--27718-- All heap blocks were freed -- no leaks are possible
--27718--
--27718-- For lists of detected and suppressed errors, rerun with: -s
--27718-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PL3PWB8:/home/hw4/code#
```

Test 3 Output With CTRL + C:



```
code > C:\main2.c > intHandler(int)
357 }
358
359 // All the workers are not ending because of the signal handler. I tried to use pthread_cancel but it didn't work.
360 void intHandler(int dummy)
361 {
362     printf("\nCTRL-C caught. Exiting...\n");
363     running = 0; // Set running to 0 to signal all threads to exit from while loop
364     pthread_cond_signal(&buffer_not_full);
365     pthread_cond_signal(&buffer_not_empty);
366     buffer.done = 1;
367     pthread_mutex_unlock(&buffer.mutex);
368     pthread_cond_broadcast(&buffer_not_empty); // Wake up all workers
369     pthread_cond_broadcast(&buffer_not_full); // Wake up all workers
370 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
--30463-- Ponchek, a memory error detector
--30463-- Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
--30463-- Using Valgrind-3.18.1 and LDBEX; rerun with -h for copyright info
--30463-- Command: ./MKP 10 10 ../testdir ../toCopy
--30463--
CTRL-C caught. Exiting...
-----STATISTICS-----
Consumers: 10 - Buffer Size: 10
Number of Regular File: 516
Number of FIFO File: 0
Number of Directory: 15
TOTAL BYTES COPIED: 1946322
Execution time: 10 seconds and 9516760 micros
--30463--
--30463-- HEAP SUMMARY:
--30463--   in use at exit: 0 bytes in 0 blocks
--30463--   total heap usage: 30 allocs, 30 frees, 534,272 bytes allocated
--30463--
--30463-- All heap blocks were freed -- no leaks are possible
--30463--
--30463-- For lists of detected and suppressed errors, rerun with: -s
--30463-- ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
root@DESKTOP-PL3PWB8:/home/hw4/code#
```


Barrier Usage:

```
// Worker function to copy files it gets from the buffer and write them to the destination directory
void *worker(void *arg)
{
    pthread_barrier_wait(&barrier); // Synchronize threads at this point to start copying files
    while (running)
    {
        FilePair file_pair = remove_from_buffer(&buffer); // Get a file pair from the buffer to copy

        if (file_pair.src[0] == '\0' && file_pair.dest[0] == '\0') // If the buffer is empty and done, exit
        {
            printf("debug\n");
        }
        pthread_barrier_wait(&barrier); // Synchronize threads at this point to finish copying files
        printf("Worker exiting...\n");
        return NULL;
    }
}
```

With this usage all the worker threads starts at the same time and they ends at the same time. This provides synchronization.