Operating Systems

6CM503

Computer GE 645 with OS Multix and its creators around

02 Assessment Brief 2025/26

Linux System Calls: Utility Design using Inter-Process Communication (IPC)

Dmitry A. Zaitsev

Module Leader

- Dr. Dmitry A. Zaitsev
- Email: d.zaitsev@derby.ac.uk, Phone: 01332592678
- Office: MS229. Recommended days for individual communication: Monday, Wednesday.
- Website: https://www.derby.ac.uk/staff/dmitry-zaitsev

Key dates and details

Assessment Detail	Assessment Information
Assessment Type:	Individual Report, Program Code
Assessment Weighting:	25%
Word count/Length:	5,000 words / 12 pages
Learning Outcomes:	1, 2
Submission Method:	Blackboard Assignment
Submission Date:	12:00 Noon UK time, 05/01/2026
Provisional Feedback Release Date:	12:00 Noon UK time, 19/01/2026

Description of the assessment

System calls represent one of the basic interfaces of OS, together with user interface and interface with computer hardware. Traditionally, system calls are implemented via a program interrupt to provide not only a call of OS kernel routine but also switching the processor context and protection mode from User to System. We can use system calls directly via a single library routine and a sequence of specified parameters; also, standard libraries of programming languages provide a series of convenient functions which represent wrappers for the corresponding system calls.

The goal of the assessment is to master system call interface of Linux and C programming using system calls, especially to implement an application as a set of communicating concurrent processes. We focus on composing programs in command line utility-like style and using such basic inter-process communication facilities of Linux as pipes, semaphores, messages, and sockets, including TCP/IP sockets.

With this assessment students master skills of using OS API of system calls, focusing on Inter-Process Communication with pipes, semaphores, messages, and sockets – for world-wide communication via the TCP/IP network. They are appointed with 2 of 4 mentioned facilities and implement a utility program as a separate process, a kind of server, with parameter transmission via given IPC. Along with the report, students submit code of the utility program, and client-server applications using given IRP.

There are 12 variants to develop and debug software using OS IPC system calls. At first, a utility program with command line interface and options is developed, like system utility interface. At second, two pairs of client-server applications are developed for a given pair of IPC within the following list: pipe, semaphore, message, sockets. The server program provides the same information processing as the utility program, parameters and results are interchanged with the client program using given IPC. The report supposed analysis of using communication facilities supplied with graphs, recommendations, and conclusions.

Basic information: Lectures 6-12.

Relationship to Programme Assessment Strategy

The second task of the module Coursework four portfolio tasks.

Attributes and Skills

The attributes and skills are listed in the following table.

	Skills	Links to useful resources
☒	Critical thinking	Analysis of historical and contemporary OSs.
☒	Communication	Working within common Linux server environment.
☒	Collaboration	Partial teamwork on the same task variants.
☒	Creative problem solving	Script, utility, and kernel loadable module design
☒	Self-direction & planning	Planning efforts and activities to satisfy task submission deadlines.
☒	Numeracy, statistics & financial literacy	Top X lists and statistics on most widespread OSs for specific application domains, awareness of financial aspects with OSs' comparative cost and reliability
☒	Digital	Students not only use but create digital technologies
☒	Resilience	Students learn how to avoid stress inflicted by with necessity to satisfy deadlines and recover after busy times
☒	Adaptability	Adaptation to using rather novel, for students, type of OS compared to officially supported by UoD MS Windows
☒	Leadership & future thinking	Developing kernel loadable modules, students think about OS of future

Assessment Content

Design programs using given OS Linux Inter-Process Communication (IPC) facilities for information exchange and synchronization between concurrent processes.

Workflow:

1. Design a utility program with command line interface (for further work with IPC) according to the task variant
2. Run and test example programs of client and server to provide service using IPC
3. Design client and server to provide service using IPC for each of two specified facilities from the list: pipes, messages, semaphores, sockets (please see your variant numbers)
4. Run and test client-server utility programs, collect evidence that they are running within Linux environment, required IPC constructs created
5. Explain in detail client-server utility program work using graphical diagrams of communication
6. Include into report information on running and testing example programs for two other IPC which were not a part of your variant

Task variants: Appendix 1.

Directions: Appendix 2.

Content of report: Appendix 3.

Example program code – module supplemental materials.

Assessment Rubric

Criteria	Excellent (70-100%)	Very good (60-69%)	Good (50-59%)	Satisfactory (40-49%)	Unsatisfactory (<40%)
Utility program design, 20%	Successful implementation of all requirements	Successful implementation of basic requirements	Successful implementation of basic requirements with minor errors	Implementation of basic requirements with some errors	Basic requirements have not been implemented
Run and test examples, 10%	Successful testing of all examples	Successful testing of all examples with minor imperfections	Successful testing of basic examples	Testing of basic examples with some errors	Basic examples have not been tested
First IPC application design, 25%	Successful implementation of all requirements	Successful implementation of basic requirements	Successful implementation of basic requirements with minor errors	Implementation of basic requirements with some errors	Basic requirements have not been implemented
Second IPC application design,	Successful implementation of all requirements	Successful implementation of basic requirements	Successful implementation of basic requirements	Implementation of basic requirements with some	Basic requirements have not been implemented

25%			with minor errors	errors	
Report writing, 20%	The report is exceptionally well-structured, clearly written, and professionally presented.	The report is well-structured and clearly written with minor errors.	The report is complete but may lack clarity or detail in some sections.	Report is poorly organized or lacks clarity. Analysis is weak or incomplete.	Report is disorganized, difficult to follow, and lacks key elements.

Assessment AI-assistance

In this assessment AI-assistance is permitted in the following ways:

- Explanation of IPC facilities
- Check of obtained results

In this assessment AI-assistance should not be used for:

- Code composition
- Writing report

It is YOUR RESPONSIBILITY to check ALL information generated by generative AI tools. Any misuse of generative AI tools could be considered ethical academic misconduct (as per Academic Regulations, Academic Misconduct, Section J2). If in doubt, please consult your module leader.

Anonymous Marking

You must submit your work using your **student number** to identify yourself, not your name. You must not use your name in the text of the work at any point. When you submit your work in Turnitin you must submit your student number within the assignment document and in the *Submission title* field in Turnitin. Guidance is available showing how to do this.

Assessment Regulations

The University's regulations, policies and procedures for students define the framework within which teaching and assessment are conducted. Please make sure you are familiar with these regulations, policies and procedures.

Appendices

Appendix 1. Variants of task

a. Utility function:

1) Concatenate two strings
2) Find a substring
3) Replace a substring
4) Compare two strings
5) Reverse a string
6) Swap two parts of string

7) Extract a substring using specified position and length
8) Remove letters specified in the second string
9) Insert a substring from the specified position
10) Remove a substring
11) Remove the inner 1/3 part of string
12) Remove the outer 1/3 parts of a string

Example: a utility to count a given string length *slen.c*

b. IPC facilities:

1) *Pipes*
2) *Messages*
3) *Semaphores*
4) *TCP/IP sockets*

- Choose the task variant number according to the file of variants

Appendix 2. Directions

2.1. Utility program

Compose a utility program with command line interface to provide the required service, implement at list 3 options (keys). Organize processing of information required by the task variant as a C language function. Work with arrays of characters and pointers directly. Use neither standard libc functions for string processing nor other libraries.

Plot: In the following tasks, we will be using the same basic code for required service (C language function) in the context of IPC.

Directions:

- Discuss with teacher and decide on the utility options.
- Run the example program and study how command line interface is implemented.
- Compose a function for the required processing of information.
- Compose the main program to implement the command line interface and the function call.
- Debug and test the program.
- Create and consider intermediate codes: after preprocessor, assembler, compiler (object code).

2.2. IPC via pipes

Exercise: *Please implement the required information processing using a pair of specialized processes – client and server; provide the information communication between the processes via Linux pipes.*

Examples: creation of processes *fork_ex.c*; communication via pipes *pipe_ex.c*

Directions:

- Use the designed utility program according to your variant.
- Compose a client that periodically inputs data and sends it to server.

- Compose a server that receives data from client, processes it, and returns the result back to client.
- It is recommended to use a starter (supervisor) process to launch client and server and to organize their communication.
- Debug and test the programs.

2.3. IPC via messages

Exercise: Please implement the required information processing using a pair of specialized processes – client and server; provide the information communication between the processes via Linux messages.

Example: communication via messages *mq_client.c, mq_server.c, mq_common.c*

Directions:

- Use the designed utility program according to your variant.
- Compose a client that periodically inputs data and sends it to server.
- Compose a server that receives data from client, processes it, and returns the result back to client.
- It is recommended to use a starter (supervisor) process to launch client and server and to organize their communication.
- Debug and test the programs.

2.4. IPC via semaphores

Exercise: Please implement the required information processing using a pair of specialized processes – client and server; provide the information communication between the processes via Linux semaphores and shared segments of memory.

Example: synchronization of dining philosophers via semaphores *ph.c, super_ph.c*; using shared segments of memory *seg_ex.c, shm_client.c, shm_server.c*

Directions:

- Use the designed utility program according to your variant.
- Compose a client that periodically inputs data and sends it to server.
- Compose a server that receives data from client, processes it, and returns the result back to client.
- It is recommended to use a starter (supervisor) process to launch client and server and to organize their communication.
- Debug and test the programs.

2.5. IPC via IP-UDP sockets

Exercise: *Please implement the required information processing using a pair of specialized processes – client and server; provide the information communication between the processes via Linux IP-UDP sockets.*

Example: communication via IP-UDP sockets *udp_client.c, udp_server.c*

Directions:

- Use the designed utility program according to your variant.
- Compose a client that periodically inputs data and sends it to server.
- Compose a server that receives data from client, processes it, and returns the result back to client.
- Use coordinated identification of client and server with a socket – IP-address and port number.
- Debug and test the programs.
- Try connection between remote computers

Additional task (optional): run and test example client-server application for stream communication based on TCP protocol: tcp_client.c and tcp_server.c

Appendix 3. Content of report

1. Utility program
   1.1. How to use the utility program
   1.2. Utility program description
   1.3. Results of the utility program (no less than 3 tests)
2. For each IPC facility specified in you variant
   2.1. How to run the client and server
   2.2. Description of client and server illustrated with graphical schemata and snippets of code
   2.3. Graphical scheme of communication for given IPC inscribed with used functions
   2.4. Results of the client-server application testing (no less than 3 tests)
3. For each IPC facility unspecified in you variant
   3.1. Results of the example client-server application testing

The report should be supplied with an archive containing (in textual files ready to run) a utility program and two pairs of client and server programs, including evidence of their testing, supplied with comments on how to use them.

○