Software Requirements Specification (SRS)

System call Microbenchmark Suite

T3

Authors : 김상재, 박성준, 이종빈, 장서연

Instructor : 진현욱 교수님

1. Introduction

1.1. Purpose

본 문서는 시스템 콜 마이크로 벤치마크 스위트의 기능과 구성요소 그리고 인터페이스를 고객과 공급자 관점에서 명시하고 상세함에 목적이 있다.
시스템 콜 마이크로 벤치마크 스위트는 각 시스템 콜마다 별개의 기준, 제약조건을 가지고 있으므로 각 시스템 콜 별로 요구사항을 상세할 수 있도록 한다. 사용되는 시스템 콜의 종류로는 Signal, Mutual exclusion (Lock), IPC (MessageQueue)가 있으며 각 시스템 콜 별로 고객의 소프트웨어에서 사용되는 해당 시스템 콜 성능 측정을 위한 측정 기준과 요구사항을 명시한다.
SRS는 기능과 구성요소 그리고 인터페이스 구현을 위해 유스케이스(Use Case), 기능 요구사항(Functional Requirement), 비기능 요구사항(Non-functional Requirement), 개념 모델링(Conceptual Modeling) 등이 참조될 수 있으며 추후에 요구사항 확인(Requirement Validation)과 요구사항 검증(Requirement Verification) 과정을 위해 이용될 수 있다.

1.2. Scope

본 문서에서 상세하는 제품을 “시스템 콜 마이크로 벤치마크 스위트(System Call Microbenchmark suite)”로 정의한다.
시스템 콜 마이크로 벤치마크 스위트는 각 종류 별 시스템 콜들의 다양한 환경 속 성능을 측정하고 개선사항에 대한 근거를 마련할 수 있어야 한다.
각 시스템 콜 마이크로 벤치마크 스위트 마다 공통적으로 다음과 같은 기능들을 가지고 있어야한다.

A. 코어(Core) 수에 따른 시스템 콜의 성능 측정
B. 프로세스(Process) 혹은 스레드(Thread) 수에 따른 시스템 콜의 성능 측정
C. 토폴로지(Topology)에 따른 시스템 콜의 성능 측정
D. 성능 측정 결과값을 CSV 파일 형식으로 저장 및 그래프 산출

시스템 콜 마이크로 벤치마크 스위트는 실제 소프트웨어를 테스트하는 것이 아닌 사용자가 원하는 시스템 콜과 미리 정해진 시스템 콜들의 실행 동작 패턴을 설정하여 테스트된다. 따라서 다음과 같은 이점을 가지게 된다.

1. 편의성
    a. 테스트 환경에 대한 오버헤드를 줄일 수 있다.
    b. 여러 환경에서 시스템 콜 성능 측정을 간단히 구성할 수 있다.
    c. CSV 파일 형식의 데이터로 부터 성능 측정의 산출물을 도출해 낼 수 있다.

2. 다양성
    a. 사용자 필요에 따라 다른 환경에서 테스트의 성능 측정 산출물을 도출해 낼 수 있다
    b. 특정 시스템 콜 에 집중해서 분석, 비교할 수 있다.

단, 시스템 콜 마이크로 벤치마크 스위트는 최대한 커널 레벨 혹은 POSIX V 라이브러리 레벨에서의 오버헤드를 측정할 수 있어야한다.

1.3. Definitions, acronyms, and abbreviations

Term	Description
1.1 벤치마크(Benchmark)	실제 기능에서 측정하고 싶은 구간의 실행 로직과 동일한 로직을 제작하여 측정하는 것.
1.2 마이크로 벤치마크(Microbenchmark)	매우 작고 특정한 코드 조각의 성능을 측정하기 위해 설계된 벤치마크.
1.3 마이크로 벤치마크 스위트(Microbenchmark suite)	마이크로 벤치마크 프로그램들의 모음.
2.1 시스템 콜(System call)	커널 영역의 기능을 사용자 모드에서 사용 가능하도록 운영체제로부터 제공받는 인터페이스
3.1 POSIX	소스코드 수준에서 애플리케이션 이식성을 지원하는 명령 해독기와 공통된 운영 프로그램들이 포함된 운영체제 인터페이스와 환경의 표준
3.2 Message queue	메시지 기반의 미들웨어로 메시지를 이용하여 여러 어플리케이션, 시스템, 서비스들을 연결해줄 때 사용되는 queue
4.1 시그널(Signal)	특정 사건이 발생했다는 사실을 프로세스 혹은 프로세스 그룹에 보내는 짧은 메시지
5.1 Mutual exclusion(Mutex)	병행 프로세스에서 프로세스 하나가 공유 자원을 사용할 때 다른 프로세스들이 동일한 일을 할 수 없도록 하는 방법
5.2 pthread_mutex	POSIX 환경에서 mutex를 지원하기 위해 제작된 함수들의 집합
5.3 세마포어(Semaphore)	두 개의 원자적 함수로 조작되는 정수 변수로서, 멀티프로그래밍 환경에서 공유 자원에 대한 접근을 제한하는 방법으로 사용
5.4 Futex	특정 상태가 true가 될때까지 기다리도록 하는 시스템 콜이다.
6.1 운영체제(OS)	시스템 하드웨어를 관리할 뿐 아니라 응용 소프트웨어를 실행하기 위하여 하드웨어 추상화 플랫폼과 공통 시스템 서비스를 제공하는 시스템 소프트웨어이다.
6.2 커널(Kernel)	컴퓨터의 운영 체제의 핵심이 되는 컴퓨터 프로그램의 하나로, 시스템의 모든 것을 완전히 통제한다

7.1 공급자(Supplier)	이 프로그램을 개발하고 제공하는 사람
7.2 고객(Customer)	이 프로그램을 사용하는 사람

1.4. References

IEEE Std 830-1998 (Reversion of IEEE Std 830-1993)

1.5. Overview

본 문서는 사용자의 요구사항과 더불어 수반되는 기능들에 대한 개략적인 구조들을 명시하고 있다.
요구사항 분석을 위해 본 문서는 구체적으로 여러 섹션으로 구성된다.
첫번째 섹션에서는 SRS의 목적과 프로젝트에서 쓰이는 주요 단어, 단축어들 그리고 프로젝트에서 다룰 내용들이 요약되어 있다.
두번째 섹션에서는 앞서 요약한 내용들을 심화하여 전체적인 프로젝트의 묘사 그리고 소프트웨어에서 사용될 기능들을 명시한다. 더불어 타겟 유저들을 명시함과 동시에 기능들과 연관된 여러 인터페이스들을 설명한다.
세번째 섹션에서는 마이크로 벤치마크 스위트의 구성요소별 기능들을 Mode를 기준으로 상세한다. 내용들은 크게 Signal, Mutual Exclusion, POSIX V message queue 로 나눠진다. 각각의 Mode들은 개별적으로 benchmark에 대한 기준들을 설명한다.
나머지 섹션에서는 부가적으로 프로젝트에서 추가로 설명해야되는 내용들에 대해서 설명한다.

2. Overall Description

2.1. Product Perspective

시스템 콜 마이크로벤치 스위트는 시스템 콜을 사용하는 고객의 소프트웨어(이하 호스트 소프트웨어)를 사용자 설정에 따라 조건과 시스템콜의 종류를 정하여 성능의 정도를 테스트하는 소프트웨어이다.
본 프로젝트를 통해 고객은 호스트 소프트웨어에서 시스템 레벨에서의 개선사항, 병목지점 유무를 확인 할 수 있으며 CSV 파일 형식을 통해 그래프를 생성하여 경향성을 분석할 수 있다.
본 프로젝트는 유저 레벨에서 호스트 소프트웨어의 성능향상 요소를 여부를 판단하는 것이 아닌 시스템 레벨에서의 개선사항을 찾는 것이다. 따라서 유저 로직에서의 종속성을 떠나 다양한 시스템에서의 최적화된 환경을 제시할 수 있다. 더불어 테스트 환경에 대한 오버헤드를 줄이고 사용자 필요에 따른 다양한 데이터값을 도출해 낼 수 있다.

2.1.1. System interfaces

2.2 에서 추가명시한다.

2.1.2. User interfaces

- 사용자는 Command Line Interface로 본 프로그램을 실행할 수 있다.
  모든 선택지는 숫자 입력으로 선택이 가능하다.

- 모든 입력지에는 숫자만 입력이 가능하다.

- 테스팅 결과는 사용자에게 CUI로 제공되며 csv 파일로 추가 저장이 가능하다.

2.1.3. Hardware interfaces

시스템 콜 마이크로 벤치마크 스위트는 x86_64 instruction set을 사용한다. 또한 결과값들을 STDOUT과 그래프로 표현하기 위한 디스플레이 표시장치로 연결된다.

2.1.4. Software interfaces

시스템 콜 마이크로 벤치마크 스위트 는 결과값들을 표시하고 파싱하기 위한 Python Interpreter 와 연결되어 구성된다. Python Interpreter에는 데이터들을 시각화하기위한 다양한 패키지가 포함되있다.
본 제품은 시스템 호출을 실행하기 위해서 Ubuntu 운영체제와 수반되는 Kernel 위에서 실행된다. 시스템 콜 마이크로 벤치마크 스위트의 지정된 설정 외 것들은 운영체제가 담당하는 자원관리 규칙에 따라 실행된다.

2.2. Product Functions

Reference	Use case	System operation
R.0.1	Select Mode	select_mode()
R.0.2	Select Topology	select_topology()
R.0.3	Input Value	input_value()
R.1.1	Select LockMode	select_lock_mode()
R.2.1	Execute whole test	exec_whole_tests()
R.2.2	Excute single test	exec_single_test()
R.3.1	Make CSV	make_csv()
R.3.2	Print Graph	print_graph()

2.3. User Characteristics

이 SW는 시스템 호출을 반복적으로 실행하는 시스템의 성능을 측정하기 위해 개발된다. 따라서 본 SW는 User level에서의 성능 측정이 아닌 Kernel level 혹은 POSIX library 범주의 성능 측정을 목표로 한다. 그러므로 본 SW는 시스템 호출과 User level 간의 인터페이스를 구현하는 유저, 혹은 기존의 시스템의 User level 최적화를 넘어서 시스템 최적화를 꾀하는 유저에게 범용적으로 쓰일 수 있다.
본 SW는 범용적 상황에 맞춰져 있기 때문에 다양한 변수, 환경들 속에서 최적화된 환경 값들을 찾는데에 사용 될 수 있다.

2.4. Constraints and Uncertainty

시스템 콜 마이크로 벤치마크 스위트는 Ubuntu 20.04 LTS, Kernel Version 5.4.0-37-generic 환경에서 작동하도록 구성되며 상위 혹은 하위 OS, Kernel 버전의 환경에서 정상적 동작을 보장하지 않는다. 따라서 사용자는 동일한 환경내에서 실행되는 것을 전제로 한다.
시스템 콜 마이크로 밴치마크 스위트의 결과값이 수치도표에 의해 개선 가능성에 대한 참조로 사용될 수 있으나 개선 부분을 직접적으로 제시, 개선의 명확한 근거가 될 수 는 없다.
시스템 콜 마이크로 벤치마크 스위트 중 프로세서 수의 대한 결과물 얻을때 싱글 코어 혹은 테스트 값보다 적은 코어 수에서 정상적인 결과값을 보장하지 않는다.

2.4.1. Hardware limitations

같은 종류의 시스템콜을 사용하는 커널, 운영체제 환경에서도 다른 Instruction set을 사용하는 CPU에서는 동일한 결과를 보장하지 않는다.

2.4.2. Regulatory policies

시스템 콜 마이크로 벤치마크 스위트는 UP(Unified Process) 소프트웨어 개발 프로세스 프레임워크로 개발된다.

2.5. Assumptions and Dependencies

시스템 콜 마이크로 벤치마크 스위트는 멀티코어 환경에서 테스트되기 위하여 멀티코어 프로세서

Python Interpreter : Python 3.8.2
OS : 20.04 LTS
Kernel Version : 5.4.0-37-generic

2.6. Apportioning of requirements

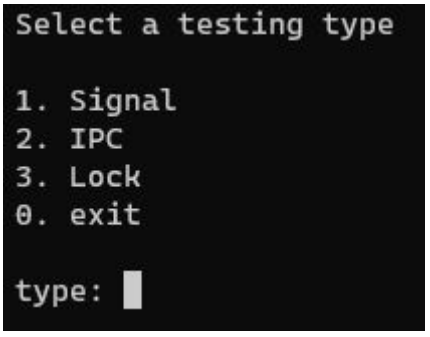
향후 버전에서 고객이 추가로 요구하는 사항을 단계적으로 추가한다.
시스템 콜 마이크로 벤치마크 스위트를 통해 성능 개선 가능성을 찾고 실제로 성능 개선점, 병목지점의 원인을 파악했을 시 문서의 Constraints and Uncertainty 를 업데이트, 기능에 성능 개선점 제안을 추가한다.
Client, Risk, Architecture driven 하게 dependency가 적은 사항부터 개발의 정도에 따라 추가로 작성한다.

3. Specific requirements

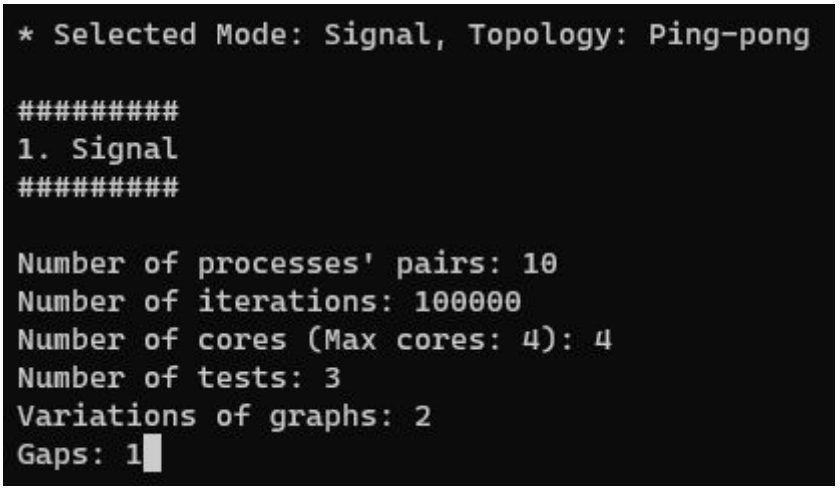
3.1. External interface requirements

3.1.1. User interfaces

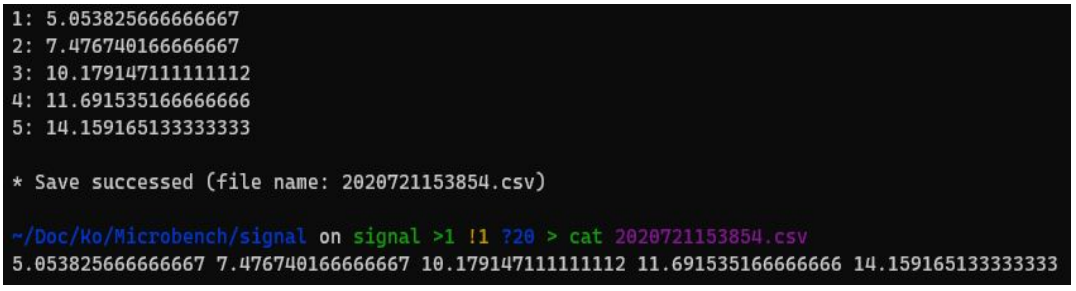
- 사용자는 Command Line Interface로 본 프로그램을 실행할 수 있다.
- 모든 선택지는 숫자 입력으로 선택이 가능하다.



- 모든 입력지에는 숫자만 입력이 가능하다.



- 테스팅 결과는 사용자에게 CUI로 제공되며 csv 파일로 추가 저장이 가능하다.



3.2. Functional requirements

3.2.1. Beginning of the program

3.2.1.1. Select Mode

Name	select_mode()
Responsibilities	초기 실행 화면으로써 시스템 콜의 종류를 선택하고 3.2.1.2의 함수로 전환한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	선택지에 존재하지 않는 값을 선택 시, 오류 메시지를 출력한다.
Output	선택된 모드의 입력창으로 전환한다.
Pre-conditions	-
Post-conditions	선택된 모드로 전환해야 한다.

3.2.1.2. Select topology

Name	select_topology()
Responsibilities	사용자가 테스팅할 토폴로지를 선택한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	1. 입력값이 토폴로지의 종류 개수 이상이거나 0미만의 정수일 경우 오류 메시지 출력
Output	현재 설정된 값들에 대한 결과를 출력한다.

3.2.1.3. Input value

Name	input_value()
Responsibilities	사용자가 설정할 수 있는 것들은 다음과 같다. 1. 코어의 개수 2. 프로세스의 개수 3. Topology의 패턴 반복 횟수 4. 테스팅 반복 횟수 사용자에게 위에 해당하는 설정을 받아야 한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	1. 프로세스들의 개수, 패턴 반복 횟수, 테스트 반복 횟수, 사용되는 코어의 수가 0이하의 숫자일 경우 오류 메시지를 출력한다. 2. 프로세스의 할당 코어 위치가 실제 물리 코어 수 미만이거나 초과한 수일 경우 오류 메시지를 출력한다.
Output	현재 설정된 값들에 대한 결과를 출력한다.

3.2.1.4. Select Lock Mode

Name	select_lock_mode()
Responsibilities	세마포어와 뮤텍스 중 사용자가 원하는 모드를 고를 수 있다.
Type	Evident
Cross References	-
Notes	-
Exceptions	선택지에 존재하지 않는 값을 선택 시, 오류 메시지를 출력한다.
Output	선택된 모드의 입력창으로 전환한다.
Pre-conditions	사용자가 function 3.2.1.1(select_mode())에서 lock인 3번을 선택했어야만 한다.
Post-conditions	선택된 모드로 전환해야 한다.

3.2.1.5. Execute whole tests

Name	exec_whole_tests()
Responsibilities	1. 설정된 값으로 테스팅을 실행한다. 2. 반환된 시간측정값들을 저장한다. 3. 그래프 생성을 위하여 x축에 해당하는 값을 변경하여 각 single test들을 진행. 4. single test들의 결과값을 취합한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	-
Output	x축 값, x축에 대응하는 시간 측정값
Pre-conditions	사용자가 테스팅을 위해 값들을 전부 설정해야한다.
Post-conditions	테스팅을 진행하고 시간 측정값들을 저장한다.

3.2.1.6. Execute single test

Name	exec_single_test()
Responsibilities	1. 테스팅을 위해 생성된 모든 프로세스들을 시작한다. 2. 모든 프로세스들로부터 시간 측정값을 받는다. 3. Topology에 따라 추가적으로 프로세스들의 시간 측정값을 계산한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	1. 프로세스 실행 도중 시스템상의 오류로 인하여 정상적인 실행이 불가능할 경우 오류를 출력하고 프로그램을 종료한다.
Output	시간 측정값
Pre-conditions	모든 프로세스들이 대기 상태여야한다.
Post-conditions	시간 측정 값을 가져야한다.

3.2.1.7. Make a csv file

Name	make_csv()
Responsibilities	테스팅 결과 값을 csv 파일로 생성하여 시간 측정 결과값들을 저장한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	-
Output	csv 파일 이름
Pre-conditions	사용자가 테스팅을 위해 값들을 전부 설정해야한다.
Post-conditions	테스팅을 진행하고 시간 측정값을 저장한다.

3.2.1.8. Print graph

Name	print_graph()
Responsibilities	사용자의 입력 여부에 따라 Graph를 출력하거나 프로그램을 종료한다.
Type	Evident
Cross References	-
Notes	-
Exceptions	-
Output	csv 파일 이름
Pre-conditions	사용자가 테스팅을 위해 값들을 전부 설정해야한다.
Post-conditions	테스팅을 진행하고 시간 측정값을 저장한다.

3.2.3. Mutual exclusion

3.2.3.1. Generate process

Name	generate_process()
Responsibilities	유저가 Input Value에서 받은 값만큼 thread를 생성한다. 1. POSIX pthread_create 를 통해 Input Value에서 설정된 thread num 만큼 thread들을 생성한다. 2. pthread_create 를 통해 thread를 생성할때 Input Value 에서 설정된 Topology에 따라 연결될 함수포인터를 지정한다. a. Global variable 토폴로지를 선택할시 ()을 함수포인터에 연결한다. b. Shared spsc 토폴로지를 선택할시 ()을 함수포인터에 연결한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	1. Input Value 로 부터 받아온 값이 음수일 경우 오류 메시지를 출력한다. 2. 시스템이 성공적으로 Thread를 생성하지 못하면 오류 메시지를 출력한다.
Output	정상적으로 설정된 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	사용자가 3.2.1.3 (input_value())에서 토폴로지와 thread의 개수를 입력해야한다.
Post-conditions	사용자가 설정한 값만큼 thread들이 생성된다.

3.2.3.2. Generate threads

Name	generate_threads()
Responsibilities	유저가 Input Value에서 받은 값만큼 thread를 생성한다. 3. POSIX pthread_create 를 통해 Input Value에서 설정된 thread num 만큼 thread들을 생성한다. 4. pthread_create 를 통해 thread를 생성할때 Input Value 에서 설정된 Topology에 따라 연결될 함수포인터를 지정한다. a. Global variable 토폴로지를 선택할시 ()을 함수포인터에 연결한다. b. Shared memory 토폴로지를 선택할시 ()을 함수포인터에 연결한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	3. Input Value 로 부터 받아온 값이 음수일 경우 오류 메시지를 출력한다. 4. 시스템이 성공적으로 Thread를 생성하지 못하면 오류 메시지를 출력한다.
Output	정상적으로 설정된 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	사용자가 3.2.1.3 (input_value())에서 토폴로지와 thread의 개수를 입력해야한다.
Post-conditions	사용자가 설정한 값만큼 thread들이 생성된다.

3.2.3.3. Set core affinity

Name	Set_core_affinity()
Responsibilities	사용자의 입력값에 따라 사용 코어의 개수를 설정한다. 1. 사용자의 입력값에 따라 sched_setaffinity()를 통해 코어의 할당량을 조절한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	1. Input Value로 부터 받아온 값이 음수일 경우 오류 메시지를 출력한다. 2. 시스템이 성공적으로 Core affinity를 설정하지 못하면 오류 메시지를 출력한다.
Output	정상적으로 설정된 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	사용자가 3.2.1.3 (input_value())에서 Core의 개수 값을 입력해야한다.
Post-conditions	사용자가 설정한 값만큼 Cpu Core가 활성화 된다.

3.2.3.4. Initialize Data structure

Name	init_data()
Responsibilities	성능측정을 위한 timespec을 각 thread에 할당하기 위하여 동적할당을 수행한다. 1. Input Value로 부터 받아온 thread의 수만큼 시작지점, 종료지점, 도착지점, 응답지점 그리고 수행지점을 생성한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	동적 할당이 수행되지 못하면 오류 메시지를 출력한다.
Output	정상적으로 초기화한 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	generate thread에서 사용자의 입력 수 만큼 thread의 개수를 생성하였어야 한다.
Post-conditions	각 timespec 변수가 동적으로 생성된다.

3.2.3.5. Initialize mutex

Name	init_mutex()
Responsibilities	세마포어 혹은 pthread_mutex를 초기화한다. 1. select_lock_mode()에서 설정된 값에 따라 pthread_mutex 혹은 세마포어를 초기화한다. a. 설정된 토폴로지에 따라 정해진 개수만큼의 세마포어 혹은 pthread_mutex를 초기화한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	세마포어 혹은 pthread_mutex의 초기화가 실패하면 오류 메시지를 출력한다.
Output	정상적으로 초기화한 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	사용자가 3.2.1.3 select_lock_mode()에서 Mutual exclusion의 종류를 입력해야한다.
Post-conditions	사용자가 설정한 Mutual exclusion의 값들이 초기화된다.

3.2.3.6. Free Data Structure

Name	free_data()
Responsibilities	성능측정을 위해 생성된 timespec을 free시켜 할당을 해제한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	해당 변수를 할당해제하지 못하는 경우 오류 메시지를 출력한다.
Output	-
Pre-conditions	3.2.2.4(init_data())에서 timespec이 만들어졌어야 한다.
Post-conditions	timespec변수가 할당 해제된다.

3.2.3.7. Free Mutex

Name	Free_mutex()
Responsibilities	세마포어의 값들을 할당 해제한다. 1. 할당된 세마포어의 개수만큼 세마포어들을 할당 해제한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	-
Pre-conditions	1. 사용자가 3.2.1.3 select_lock_mode()에서 Mutual exclusion의 종류를 입력해야한다. 2. 할당해제하려는 세마포어의 값들이 사용이 모두 종료된 상태가 되어야 한다.
Post-conditions	할당되었던 세마포의 값들이 모두 할당 해제된다.

3.2.3.8. Access global variable

Name	access_global_varable()
Responsibilities	mutual exclusion의 토폴로지 중 하나인 Access global variable 토폴로지를 실행한다. 1. 지정된 thread 개수만큼 글로벌 변수 g에 대하여 다음과 같은 작업들이 진행된다. a. pthread_mutex_lock 혹은 sem_wait으로 보호된 Critical Section을 Input Value에서 지정된 반복 횟수을 도달할때까지 접근을 시도한다. b. access_global_varialbe()의 동작을 수행하는 thread는 critical section에 진입하지 못하면 blocking 상태로 process life cycle이 전환된다. c. Critical Section을 점유하고 있는 thread가 Critical Section을 벗어날때 sem_post 혹은 pthread_mutex_unlock을 호출하여 blocking된 thread들을 ready상태로 전환한다. d. Scheduler에 의해 실행될 thread가 선택되며 1 ~ 4 과정들이 반복된다. 2. 실행 중인 모든 thread에 대하여 a ~ d 가 지정된 반복 횟수를 충족했을때 acess_global_variable()의 소요시간들을 thread 별로 출력한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	총 소요 시간을 출력한다.
Pre-conditions	generate_thread(), set_core_affinity
Post-conditions	사용자의 입력에 대한 테스팅 환경을 설정한다.

3.2.3.9. SPSC Producer

Name	spsc_producer
Responsibilities	생산자의 역할로 아이템을 만들어 임계영역에 아이템을 넣는다. 1. spsc_producer는 다음과 같은 순서로 지정된 패턴 반복횟수에 도달때 까지 작동한다. a. 임계 영역이 비었는지 확인한다. b. 임계 영역에 접근할 수 있는지 확인한다. c. 아이템을 임계 영역에 넣는다. d. 임계 영역 접근에 대한 mutex값을 변경한다. e. 임계 영역에 아이템이 있다고 표시한다. 2. 1 에서의 시간을 측정하여 출력한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	총 소요 시간을 출력한다.
Pre-conditions	임계 영역 접근에 대한 mutex가 설정되어 있어야 한다. 임계 영역을 설정하여야 한다. 임계 영역이 비어있어야 한다.
Post-conditions	임계 영역에 아이템이 입력된다. 임계 영역에 대한 접근 및 상태에 대한 mutex가 갱신된다.

3.2.3.10. SPSC Consumer

Name	scsp_consumer
Responsibilities	소비자의 역할로 아이템을 만들어 임계영역에 아이템을 비운다. 1. 임계 영역에 아이템이 있는지 확인한다. 2. 임계 영역에 접근할 수 있는지 확인한다. 3. 임계 영역을 비운다. 4. 임계 영역 접근에 대한 mutex값을 변경한다. 5. 임계 영역이 비었다고 표시한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	-
Pre-conditions	임계 영역 접근에 대한 mutex가 설정되어 있어야 한다. 임계 영역에 아이템이 있어야 한다.
Post-conditions	임계 영역이 비워진다. 임계 영역에 대한 접근 및 상태에 대한 mutex가 갱신된다.

3.2.3.11. Shared SPSC Consumer

Name	shared_spsc_consumer()
Responsibilities	mutual exclusion의 토폴로지 중 하나인 shared spsc 토폴로지를 실행한다. 1. 중복된 임계 영역에 아이템이 있는지 확인한다. 2. 임계 영역에 접근할 수 있는지 확인한다. 3. 임계 영역을 비운다. 4. 임계 영역 접근에 대한 mutex값을 변경한다. 5. 중복된 임계 영역이 비었다고 표시한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	총 소요 시간을 출력한다.
Pre-conditions	generate_thread(), set_core_affinity
Post-conditions	사용자의 입력에 대한 테스팅 환경을 설정한다.

3.2.3.12. Shared SPSC Producer

Name	shared_spsc_producer
Responsibilities	mutual exclusion의 토폴로지 중 하나인 Access global variable 토폴로지를 실행한다. 1. 지정된 thread 개수만큼 글로벌 변수 g에 대하여 다음과 같은 작업들이 진행된다. a. 중복된 임계 영역이 비었는지 확인한다. b. 임계 영역에 접근할 수 있는지 확인한다. c. 아이템을 임계 영역에 넣는다. d. 임계 영역 접근에 대한 mutex값을 변경한다. e. 중복된 임계 영역에 아이템이 있다고 표시한다. 2. 실행 중인 모든 thread에 대하여 a ~ e 가 지정된 반복 횟수를 충족했을때 shared_spsc_producer의 소요시간들을 thread 별로 출력한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	총 소요 시간을 출력한다.
Pre-conditions	generate_thread(), set_core_affinity
Post-conditions	사용자의 입력에 대한 테스팅 환경을 설정한다.

3.2.5. Message Queue

3.2.5.1. Initialize Message Queue Test Structure

Name	init_mqt()
Responsibilities	사용자가 Input Value에서 받은 테스트에 사용될 프로세스 개수, 코어 개수, 패턴 반복 횟수, 테스트 횟수, 토폴로지를 특정 구조체에 입력하여 초기화한다.
Type	Hidden
Cross References	R.3.2.1.3
Notes	-
Exceptions	1. Input Value 로 부터 받아온 값이 음수일 경우 -1을 리턴한다.
Output	정상적으로 초기화한 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	사용자가 3.2.1.3 (input_value())에서 프로세스 개수, 코어 개수, 패턴 반복 횟수, 테스트 횟수, 토폴로지를 입력해야 한다.
Post-conditions	사용자가 설정한 값만큼 구조체가 초기화 된다.

3.2.5.2. Generate Process

Name	generate_process()
Responsibilities	메시지 큐 테스트 구조체의 프로세스 개수 변수 값 만큼 테스트를 진행할 프로세스를 값 만큼 생성한다.
Type	Hidden
Cross References	R.3.2.3.1, R3.2.3.3
Notes	-
Exceptions	1. 시스템이 성공적으로 프로세스를 생성하지 못하면 -1을 리턴한다. 2. 구조체의 프로세스 변수 값이 음수일 경우 -1을 리턴한다.
Output	프로세스가 정상적으로 생성된 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	메시지 큐 테스트 구조체가 초기화 되어야한다.
Post-conditions	메시지 큐 테스트 구조체의 프로세스 변수 만큼 프로세스가 생성된다.

3.2.5.3. Set Core Affinity

Name	set_core_affinity()
Responsibilities	메시지 큐 테스트 구조체의 코어 개수 변수 값 만큼 프로세스가 사용할 코어의 개수를 설정한다. 2. 메시지 큐 테스트 구조체의 코어 개수 변수 값 만큼 sched_setaffinity()를 통해 코어의 할당량을 조절한다.
Type	Hidden
Cross References	R.3.2.3.1
Notes	-
Exceptions	1. 메시지 큐 테스트 구조체의 코어 개수 변수 값이 음수일 경우 -1을 리턴한다. 2. 시스템이 성공적으로 core affinity를 설정하지 못하면 -1을 리턴한다.
Output	core affinity가 정상적으로 설정된 경우 0을 리턴하고 아닌 경우 -1을 리턴한다.
Pre-conditions	메시지 큐 테스트 구조체가 초기화 되어야한다.
Post-conditions	메시지 큐 테스트 구조체의 코어 변수 만큼 프로세스의 core affinity가 설정 된다.

3.2.5.4. Initialize Message Queue Attribute

Name	init_mq_attr()
Responsibilities	메시지 큐 속성 구조체를 시스템이 정의한 임의의 값으로 초기화 한다.
Type	Hidden
Cross References	R.3.2.3.2
Notes	-
Exceptions	-
Output	-
Pre-conditions	테스트를 위해 토폴로지가 선택되고 프로세스가 생성되어야 한다.
Post-conditions	프로세스마다 메시지 큐 속성 구조체가 초기화 된다.

3.2.5.5. Initialize Message Queue Data

Name	init_mq_data()
Responsibilities	메시지 큐 데이터 구조체를 시스템이 정의한 임의의 값으로 초기화 한다.
Type	Hidden
Cross References	R.3.2.3.2
Notes	-
Exceptions	-
Output	-
Pre-conditions	테스트를 위해 토폴로지가 선택되고 프로세스가 생성되어야 한다.
Post-conditions	프로세스마다 메시지 큐 데이터 구조체가 초기화 된다.

3.2.5.6. Message Queue Send

Name	my_mq_send()
Responsibilities	인자로 명시된 메시지 큐 디스크립터에 시스템이 설정한 임의의 메시지를 보낸다.
Type	Hidden
Cross References	R.3.2.3.4, R.3.2.3.5
Notes	-
Exceptions	1. 메시지 큐 데이터를 정상적으로 보내지 못했다면 -1을 리턴한다.
Output	메시지 큐에 데이터를 정상적으로 보내면 0을 리턴하고 보내지 못하면 -1을 리턴한다.
Pre-conditions	테스트를 위해 토폴로지가 선택되고 프로세스가 생성되어야 한다. 메시지 큐 속성과 데이터가 초기화 되어야 한다.
Post-conditions	타겟 메시지 큐에 데이터를 성공적으로 저장한다.

3.2.5.7. Message Queue Receive

Name	my_mq_receive()
Responsibilities	인자로 명시된 메시지 큐 디스크립터에서 데이터를 받아온다.
Type	Hidden
Cross References	R.3.2.3.4, R.3.2.3.5
Notes	-
Exceptions	1. 메시지 큐 데이터를 정상적으로 받지 못했다면 -1을 리턴한다.
Output	메시지 큐에 데이터를 정상적으로 받으면 0을 리턴하고 받지 못하면 -1을 리턴한다.
Pre-conditions	테스트를 위해 토폴로지가 선택되고 프로세스가 생성되어야 한다. 메시지 큐 속성과 데이터가 초기화 되어야 한다.
Post-conditions	타겟 메시지 큐로부터 데이터를 성공적으로 받아온다.

3.2.6. Signal

3.2.6.1. Signal Test Main

Name	sig_test
Responsibilities	사용자가 테스팅을 위한 설정 값을 가져와서 테스팅을 진행하고 시간 측정값을 반환한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	시간 측정값
Pre-conditions	사용자가 Signal 모드에 대한 테스팅 설정값을 설정하고 실행하여야 한다.
Post-conditions	-

3.2.6.2. Signal Test Setting Attribute

Name	sig_test_setattr
Responsibilities	테스팅 환경을 위하여 설정값을 적용한다. 1. 프로세스 (혹은 프로세스 쌍)의 개수 2. 한 테스트 내부에서 한 topology 패턴 반복 횟수 3. 테스팅이 실행되는 코어의 개수
Type	Hidden
Cross References	-
Notes	-
Exceptions	1. 프로세스의 개수, topology 패턴 반복 횟수, 코어 수, 반복 횟수에 1 이상의 정수가 입력되지 않았을경우 예외
Output	성공일 경우 0, 실패일 경우 -1
Pre-conditions	-
Post-conditions	-

3.2.6.3. Signal Test Initialization

Name	sig_test_init
Responsibilities	테스팅 환경값에 따라 테스팅을 위한 초기 설정을한다. 1. 프로세스들을 생성 a. topology에 해당하는 함수 실행 b. 생성 시 topology 패턴 반복 횟수를 적용 2. 코어에 생성된 프로세스들을 할당
Type	Hidden
Cross References	-
Notes	-
Exceptions	-
Output	성공일 경우 0, 실패일 경우 -1
Pre-conditions	사용자가 테스팅 환경에 대한 값을 입력해야한다.
Post-conditions	모든 프로세스들이 대기 상태여야 한다.

3.2.6.4. Signal Test

Name	sig_test_exec()
Responsibilities	1. 테스팅을 위해 생성된 모든 프로세스들을 시작한다. 2. 모든 프로세스들로부터 시간 측정값을 받는다. 3. Topology에 따라 추가적으로 프로세스들의 시간 측정값을 계산한다.
Type	Hidden
Cross References	-
Notes	-
Exceptions	1. 프로세스 실행 도중 시스템상의 오류로 인하여 정상적인 실행이 불가능할 경우 오류를 출력하고 프로그램을 종료한다.
Output	시간 측정값
Pre-conditions	모든 프로세스들이 대기 상태여야한다.
Post-conditions	시간 측정 값을 가져야한다.

3.3. Performance requirements

3.3.1. 프로세스들은 반드시 1개 이상의 코어에 할당되어있어야 한다.
3.3.2. 총 생성되는 프로세스들은 2개 이상 **N개** 이하여야 한다.

5. Prototype

5.1. Sample Scenarios

1) 테스팅할 시스템 콜을 선정합니다.

```
Select a testing type

1. Signal
2. IPC
3. Lock
0. exit

type: 1

* Selected Mode: Signal
```

2) 테스팅할 Topology를 선택합니다.

```
Select topology

1: Ping-pong
0. exit

Topology: 1

* Selected Mode: Signal, Topology: Ping-pong
```

3) 프로세스 혹은 프로세스 쌍의 개수를 입력합니다.

```
##########
1. Signal
##########

Number of processes' pairs: 10
```

4) Topology의 패턴 반복 횟수를 설정합니다.

```
Number of iterations: 100000
```

5) 프로세스들이 실행될 Core의 개수를 설정합니다.

```
Number of cores (Max cores: 4): 4
```

6) 테스팅 횟수를 설정합니다. 성능 측정 결과는 테스팅의 평균값이 됩니다.

```
Number of tests: 3
```

7) 그래프의 종류를 설정합니다.

```
Variations of graphs: 2
```

8) 성능의 요인이 되는 x축의 간격을 설정합니다.

```
Gaps: 1
```

9) 입력값들을 확인합니다.

```
* Test Attribute

0. Number of processes: 10
1. Number of iterations: 100000
2. Number of cores: 4
3. Number of tests: 3
4. Graph type: 2
5. Number of gap: 1

Confirm? (Y/N):
```

10) 실험을 진행합니다.

```
[-] Testing : Total pairs: 10, Current pairs: 2, Current iterations: 1
```

11) 실험 결과값이 출력 됩니다.

```
1: 511.53240166666666
2: 695.3610001666666
3: 1136.4388466666667
4: 969.3274660000001
5: 1239.4369398000001
6: 1451.1570031666668
7: 1764.011125047619
8: 2023.2187185
9: 2297.4934606666666
10: 2487.4833473999993

* Save successed (file name: 2020721224559.csv)
```

12) csv 파일이 생성됩니다.

```
~/Doc/Ko/Microbench/signal on signal !1 ?2 > cat 2020721224559.csv
511.53240166666666 695.3610001666666 1136.4388466666667 969.3274660000001 1239.4369398000001 1451.1570031666668 1764
.011125047619 2023.2187185 2297.4934606666666 2487.4833473999993
```

13) 그래프 출력로도 출력합니다.

6. References

김주호, 조중연, 진현욱. (2017). 공유메모리를 이용한 향상된 메시지 기반 시스템 호출. 한국정보과학회 학술발표논문집, 1692–1692.

엄준용, 조중연, 진현욱. (2017). 네트워크 성능향상을 위한 시스템 호출 수준 코어 친화도. 정보과학회 컴퓨팅의 실제 논문지, 23(1), 80–84.

Kim, J., Cho, J.-Y., & Jin, H.-W. (2018). Application-transparent scheduling of socket system calls on many-core systems. Ancs, 174–176.

박주광, 김주호, 조현철, 진현욱. (2017). 드론 비행제어 프로그램을 위한 계층적 ARINC 653의 파티션 내 통신 구현. 한국정보과학회 논문지 제44권 제7호

7. Index

Introduction 2
- Purpose 2
- Scope 2
- Definitions, acronyms, and abbreviations 3
- References 4
- Overview 4

Overall Description 5
- Product Perspective 5
  - System interfaces 5
  - User interfaces 5
  - Hardware interfaces 5
  - Software interfaces 5
- Product Functions 6
- User Characteristics 6
- Constraints and Uncertainty 6
  - Hardware limitations 7
  - Regulatory policies 7
  - Assumptions and Dependencies 7
- Apportioning of requirements 7

Specific requirements 8
- External interface requirements 8
  - User interfaces 8
- Functional requirements 9
  - Beginning of the program 9
    - Select Mode 9
      - Select topology 9
      - Input value 10
      - Select Lock Mode 10
      - Execute whole tests 11
      - Execute single test 11
      - Make a csv file 12
      - Print graph 12
  - Mutual exclusion 13
    - Generate process 13
      - Generate threads 14
      - Set core affinity 15
      - Initialize Data structure 15
      - Initialize mutex 16

Free Data Structure 16
Free Mutex 17
Access global variable 18
SPSC Producer 19
SPSC Consumer 20
Shared SPSC Consumer 21
Shared SPSC Producer 22
Message Queue 23
Initialize Message Queue Test Structure 23
Generate Process 23
Set Core Affinity 24
Initialize Message Queue Attribute 24
Initialize Message Queue Data 25
Message Queue Send 25
Message Queue Receive 26
Signal 26
Signal Test Main 26
Signal Test Setting Attribute 26
Signal Test Initialization 27
Performance requirements 28
Prototype **29**
Sample Scenarios 29
References **31**
Index **32**