졸업작품/논문 중간보고서

2017 학년도 제 2 학기

제목	다중 사용자 환경에서의 Window Decoration 방식에 따른 성능 분석	○ 작품( ) 논문(✓)
평가등급	지도교수 수정보완 사항	팀원 명단
B A, B, F중 택1 (지도교수가 부여)	○ 다중 사용자 환경을 위한 실험 계획 방법이 미비함 ○ 실험에 사용된 워크로드에 대한 분석이 필요 ○ Window Decoration 방식에 따른 단순 성능 측정이 아닌 심층적 분석 필요	황인규 (인) (학번: 2012310466)

2017 년 9 월 20 일

지도교수 : 엄 영 익 서명

■ 요약

기존의 리눅스 시스템은 단일 사용자가 사용하는 컴퓨팅 환경에 개발되어왔다. 하지만 대형 멀티 터치스크린과 같은 다중 입력을 위한 폼팩터(Form Factor)가 등장하면서 기존의 리눅스 환경과는 다른 새로운 환경이 발생하게 되었다. 이는 사용자가 다중으로 증가하였을 뿐만 아니라 마우스와 같은 단일 포인터 이벤트가 아닌 Multi-touch 이벤트가 발생하는 환경으로써, 단일 사용자가 사용하는 환경의 CPU 사용 특성과 입출력의 특성뿐만 아니라 Window Manager가 이벤트를 처리하는 특성도 변화하였다. 이렇게 변화된 다중 사용자 환경을 위해, CPU 프로세스 스케줄러와 파일 입출력 스케줄러에 관한 다양한 연구들이 진행 되었다. 하지만, Window Manager는 단일 사용자 환경에서 다중 사용자 환경으로 변화하였음에도 많은 연구가 이루어지지 않아, 다중 사용자 환경에서의 최적화가 많이 진행되지 않았다.

다중 사용자 환경에서는 여러 프로세스와 여러 사용자가 상호작용하기 때문에 사용자들에게 낮은 응답시간(Latency)를 제공하는 것이 중요하다. 모든 사용자의 응답시간을 최소화해 성능을 높여주기 위해서는 테이블 탑 디스플레이와 같은 환경에서 빈번하게 일어날 수 있는 Window Decoration 이벤트를 처리하는 성능이 중요하다고 판단하였다.

이에 본 논문에서는 다중 사용자 환경에서 현재 쓰이고 있는 두 개의 Window Manager의 성능을 비교 분석하였다. 두 Window Manager는 서로 다른 Window Decoration 방식을 채택하고 있기 때문에 Window Manager의 성능 평가를 통해 다중 사용자 환경에서 더 나은 성능을 위한 이벤트 처리 방식을 알아보고자 했다. 두 Window Manager는 각각 CSD(Client Side Decoration)를 사용하는 Wayland와 SSD(Server Side Decoration)를 사용하는 X Window를 선택하여 실험을 진행하였다. 각각의 Window Manager에 여러 Window Decoration 이벤트를 실행하여 성능을 측정하고 분석하였다. 그 결과, Task Clock과 Context Switch는 Wayland가 더 좋은 성능을 나타내는 것을 확인하였다. 이 논문을 바탕으로 KDE가 개발 중인 DWD(Dynamic Window Decoration) 방식에 대해서도 성능 측정 및 분석을 할 예정이다.

■ 서론

□ 제안배경 및 필요성

최근 테이블 탑 디스플레이 등 다중 사용자가 동시에 사용할 수 있는 기기들이 증가하고 있다. 이러한 디스플레이의 GUI 환경에서 각 윈도우를 관리할 목적으로 만들어진 소프트웨어를 Window Manager라고 한다. 화면상에 나타나는 여러 가지 창(Window) 의 크기, 이동, 배열 등을 관리하여 보다 효율적으로 모니터에 출력하게 해준다. 서버와 클라이언트 사이의 규약인 Window Protocol, 클라이언트에 요구에 따라 화면에 출력해주는 Compositor, Window Decoration을 담당하는 Shell, 이 3가지로 이루어져 있다. 대표적으로 X Window와 Wayland Compositor가 있고 각각 Window Decoration Protocol인 X와 Wayland를 적용한 Window Manager이다. 현재 이러한 Window Manager들은 단일 사용자 환경에서 최적화되어있는 것이고 다중 사용자 환경에서는 단일 사용자 환경과는 다른 윈도우에 대한 이벤트들이 발생하기 때문에 다중 사용자 환경에 맞게 수정이 필요하다. 그림 1은 다중 사용자 환경에서 Window Manager의 서버에 발생하는 이벤트를 설명한 그림이다. Wayland Compositor와 X Window의 가장 큰 차이점으로는 Window Decoration의 방식에 있다. X Window는 서버에서 Window Decoration을 처리하는 SSD 방식을 사용하고 Wayland Compositor는 서버가 처리 하던 Window Decoration을 클라이언트에서 처리하는 방식인 CSD을 사용한다. Window Decoration의 방식인 CSD, SSD 두 방식을 어떻게 적용하느냐에 따라 다중 사용자 환경에서의 성능상의 차이를 보여줄 수 있다. CSD 방식을 선호하는 Wayland와 기존의 SSD 방식을 선호하는 X Window 사이에 어떤 방식이 더 효율적인가에 대한 논쟁이 있어왔다.

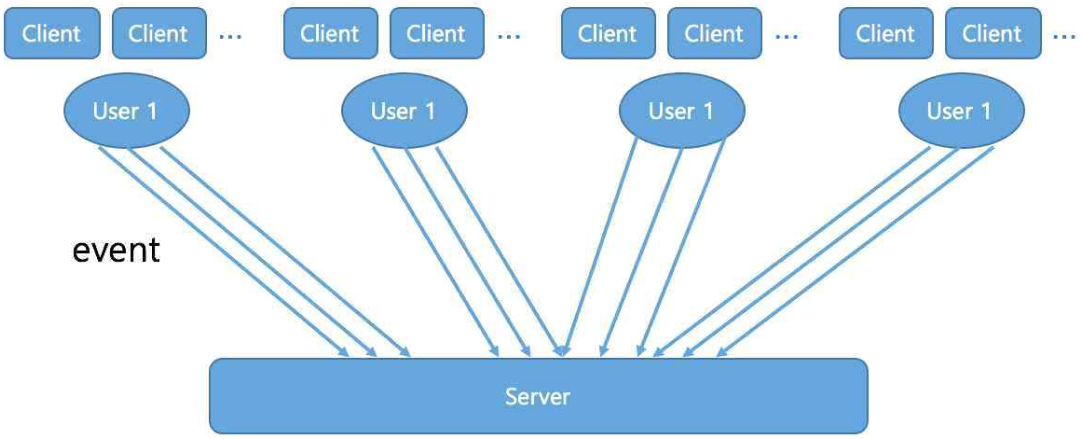

그림 1 다중 사용자 환경에서 발생하는 이벤트

단일 사용자 환경의 경우, Window Manager 서버에 요청을 보내는 클라이언트의 수가 단일 사용자만 존재하기 때문에 Window Manager가 감당할 수 있을 만큼으로 유지된다. 따라서 이미 Window Manager가 최적화되어있는 단일 사용자 환경에서는 사용자가 느끼는 디스플레이 응답성은 어떤 Window Decoration 방식을 채택한다하여도 매우 높게 측정되어진다. 하지만 다중 사용자 환경은 여러 사용자의 여러 프로세스가 서로 상호작용을 해야 하는 복잡한 상황에 놓여있다. 예를 들어 테이블 탑 디스플레이에서 여러 사용자가 수많은 프로그램을 실행하고 한 사용자가 다른 사용자에게 자신이 실행한 프로그램을 넘겨준다든지 서로 다른 사용자가 윈도우(클라이언트)에 대한 창 크기 변화, 최대화, 최소화, 종료에 따른 이벤트들을 동시에 발생시킨다든지 하는 Window Manager가 처리해야 하는 이벤트가 증가하게 된다. 따라서 다중 사용자 환경에서 어떤 Window Decoration 방식이 더 효율적인지 알 필요성이 있다.

□ 졸업작품/논문의 목표

본 논문에서는 서로 다른 Window Decoration 방식을 채택한 Window Manager인 X Window와 Wayland Compositor가 다중 사용자 환경에서 빈번하게 혹은 동시에 발생할 수 있는 Window Manager가 처리해야 하는 이벤트들에 대한 memory나 CPU 등 리소스를 더 효율적으로 사용하고 있는지, 사용자 응답시간이 짧은지에 대해 알아보고자 한다.

다중 사용자 환경에서는 모든 사용자에 대해 대화형 프로세스의 응답시간을 줄여 응답성을 높이는 것이 중요하다. 그 일환으로 다중 사용자 환경에서 최적화된 소프트웨어 플랫폼이나 입출력 스케줄러에 대한 연구는 활발히 진행 중에 있다. 다중 사용자 환경에 대한 최적화 방안으로 단일 스레드로 동작하는 Window Manager를 프로세서마다 스레드를 하나씩 만들어서 멀티 스레드로 동작시키는 연구가 있다 [1]. 다중 사용자로 인해 너무 많은 이벤트가 쌓여 단일 스레드인 Window Manager가 병목이 되는 것을 막기 위해 병렬화를 통해 멀티 스레드로 만들어 모든 사용자에 대한 대화형 프로세스의 응답성을 높인 연구이다. 또 다중 사용자 환경에 최적화된 입출력 스케줄러에 대한 연구로 CFQ 입출력 스케줄러를 기반으로 하는 CFQS 스케줄러 연구가 있다 [2]. CFQS 스케줄러는 프로세스 별 읽기 요청 큐와 쓰기 요청 큐를 분리하여 관리하고, 읽기 요청에 대해 높은 우선순위를 제공하여 읽기 요청에 대한 처리시간을 최소화시켜 사용자의 응답시간를 최소화시켰다.

다중 사용자 환경이 증가함에 따라 다중 사용자 환경에 최적화된 방식에 대한 필요성이 증가했다. 본 연구에서는 다중 사용자 환경에 맞는 Window Decoration 방

식이 무엇인지 알아보고 사용자의 응답시간를 최소화하는 방식을 찾아 제안하고자 한다.

□ 졸업작품/논문 전체 overview

다중 사용자 / Multi-Touch 환경이 증가하고 있고 이에 따라 다중 사용자 환경에 최적화된 플랫폼의 필요성 또한 함께 증가하고 있다. 다중 사용자 환경에서는 각 사용자 모두에게 좋은 응답시간을 보여주어야 한다. 이를 위해 파일 입출력 성능 향상을 위해 입출력 스케줄러를 수정한다든지 혹은 모든 사용자에게 공평한 스케줄링을 위한 CPU스케줄러 수정을 하는 일들을 해왔다.

본 논문에서는 다중 사용자 환경에서 Window Manager의 Window Decoration 방식에 따른 성능 차이를 실험해보고자 한다. 다중 사용자 환경에서 자주 발생할 수 있는 이벤트들에 대한 벤치마크 테스트를 진행하였다. Window를 새로 생성하거나 종료시키는 이벤트, 크기를 변화시키고 이동시키는 이벤트들에 대해 진행하였다. 서버 사이드에서 Window Decoration을 처리하는 방식인 SSD는 X11 / openbox로 환경을 구성하였고 클라이언트에서 Window Decoration을 처리하는 CSD는 Wayland / Weston으로 환경을 구성하였다.

이 후 섹션들은 Window Manager 관련 연구들과 실험을 진행한 환경들에 대한 설명이 이어진다. 실험 환경에 대한 정의와 벤치마크 테스트 프로그램 / 테스트 옵션 선정 기준들에 대해 설명한다. 단일 사용자 환경에서의 벤치마크 테스트 결과를 Window Decoration 방식에 따라 2가지를 비교한 결과를 그래프로 보여주고 앞으로의 연구 과제에 대해 설명한다. 마지막으로 졸업 논문을 진행하면서 겪었던 경험들에 대한 내용으로 마무리한다.

■ 관련연구

□ X Window / Wayland Compositor

X Window는 1984년부터 개발되기 시작한 윈도우 시스템이다 [3]. X 윈도우 시스템 프로토콜을 기반으로 하는 클라이언트와 서버 모델의 네트워크 지향 윈도우 시스템이다. X Window는 개발이 진행한지 오래 지났기 때문에 예전에 작성되었던 함수들 중에 X Window에 포함은 되어있지만 지금은 쓰지 않은 함수들도 포함되어있다. 기본적으로 SSD 방식을 택해서 자칫 서버 무겁게 한다. 프로그램들이 윈도우 창을 띄울 때 클라이언트가 되어 서버에 여러 가지 이벤트에 대한 요청을 보내고 서버는 해당 요청을 처리하고 그것을 컴포지터에 전달해주고 컴포지터가 화면에 출력하는 과정을 갖는다. 그림 2는 X Window의 구성도이다.

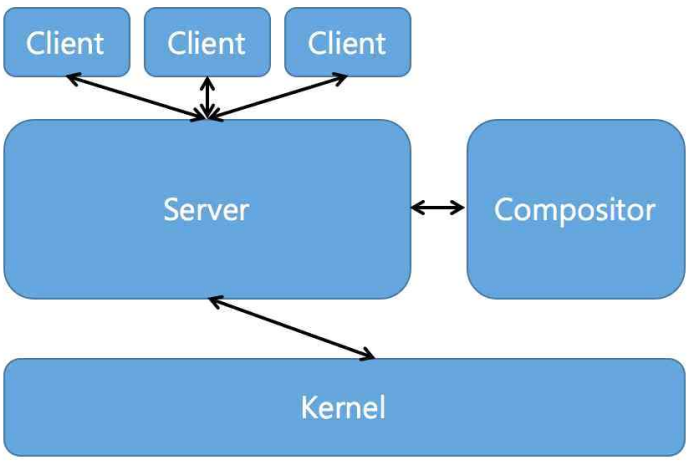

그림 2 X Window의 구성

Wayland Compositor는 오래 되고 무거운 X Window를 대체하고자 만들어진 프로토콜인 Wayland의 Window Manager이다 [5]. 서버를 가볍게 유지하는 것이 이 프로토콜의 목적이기 때문에 서버의 Window Decoration 작업을 뺏어오는 CSD 방식을 선호한다. X Window와 달리 Wayland Compositor는 컴포지터 자체가 서버의 역할을 맡고 있는 구조이다. 그림 3은 Wayland의 구성도이다.

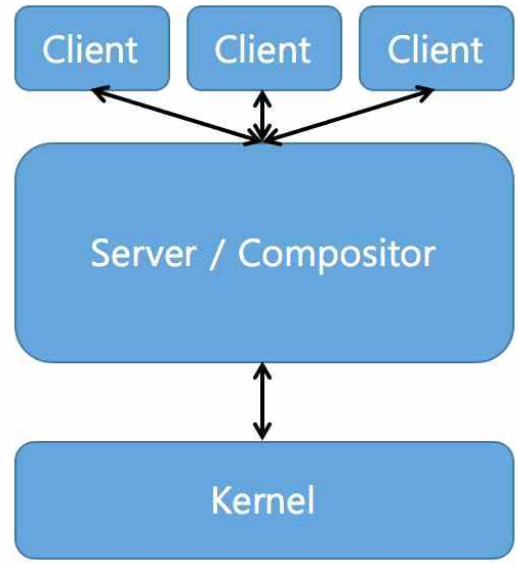

그림 3 Wayland Compositor의 구성

□ CSD 방식 / SSD 방식

먼저 Window Decoration이란 Windowing System에서 클라이언트(윈도우 창)의 title bar나 border를 수정하는 일을 일컫는다. 윈도우를 최소화, 최대화, 종료, 사이즈 조절을 하는 등의 일들을 말한다. 이러한 Window Decoration을 수행하는 방식에 SSD와 CSD이 있는 것이다.

X Window가 사용하고 있는 SSD의 경우, Window Decoration을 서버 사이드에서 진행하는 방식으로, 서버에 부하가 커지는 문제가 있지만, 모든 윈도우의 title bar를 서버에서 그리기 때문에 똑같이 일관성 있게 관리하기에 좋다. 클라이언트가 종료 이벤트 요청을 보냈을 때 클라이언트에 어떤 오류가 발생했는지에 상관없이 종료 이벤트를 처리할 수 있다.

Wayland Compositor가 사용하는 방식인 CSD는 서버가 하던 Window Decoration을 클라이언트가 직접 하게 하는 방식이다. 클라이언트가 서버의 일을 가져오기 때문에 서버를 더 가볍게 사용할 수 있지만, 클라이언트들끼리 일관성 있는 title bar를 가지려는 노력이 필요하다. CSD 는 윈도우 프레임을 클라이언트가 그리기 때문에 최소/최대/종료/사이즈 조절 등의 이벤트를 클라이언트가 확인해서 서버에게 요청해야 한다. 따라서 클라이언트가 어떤 오류에 의해 해당 이벤트를 처리하지 못 하면 정상적인 이벤트 처리가 힘들어진다.

□ X Window와 Wayland Window Manager의 성능 비교 분석

대표적인 리눅스 Window Manager 프로토콜로는 Wayland와 X window가 있다. Wayland가 무거운 X window 서버의 대안으로 제시되었지만 점유율 면에서 여전히 X window를 넘어서지 못하고 있다. 이 논문에서는 Wayland가 쓰이지 않는 원인을 분석하기 위해 Wayland와 X window의 성능 차이 실험을 하였다 [7]. 실험 환경은 Ubuntu 16.04 LTS 64-bit 리눅스 커널 4.4 버전을 사용하였고 CPU는 Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz를 사용하였다. RAM는 8GB 짜리를 사용하였다. 실험 과정에서의 CPU와 memory 사용량 측정을 위해서 top이라는 명령어를 사용하였다. 4가지의 옵션(dot, circle, rect, strap)으로 동시에 3개의 x11perf를 동시에 실행해서 실험을 진행하였다. 실험 결과, 레지던트 메모리는 8GB RAM에서 평균 460KB (0.005%) 만큼 Wayland가 적게 사용하였고 CPU의 경우, 1% 미만의 인식할 수 없는 정도의 차이를 나타내었다. 결론적으로 기존의 X window를 Wayland로 바꿀만한 성능상의 이점을 크게 보이지 못하였다. 이 실험은 다중 사용자 환경을 고려하지 않은 단일 사용자 환경에서의 실험이었다. 본 연구에서는 간단한 윈도우 디스플레이가 아닌 다중 사용자 환경에서 빈번하게 발생할 수 있는 이벤트에 대한 실험을 진행할 예정이다.

□ Window Manager for Multi-Touch Interaction

Multi-Touch 환경이 곳곳에 등장하기 시작하면서 그 환경에 맞는 어플리케이션이 하나하나 만들어지기 시작하였다. Multi-Touch 환경에서는 이러한 어플리케이션뿐만 아니라 이전에 존재했던 기존 어플리케이션에 대해서도 이상 없이 제공할 수 있어야 한다. 이 논문에서는 Multi-Touch 환경에서 현재 Multi-Touch가 아닌 환경에 적합하게 만들어진 어플리케이션과 Multi-touch에 맞게 제작한 어플리케이션을 어떻게 동시에 제공할 것인지, 그리고 마우스, 터치펜, 터치와 같이 다양한 input 양상에 대해 어떻게 처리할 것인지에 대해 이야기하고 있다 [5].

이 논문에서는 이미 존재하고 있던 기존 어플리케이션은 Legecy 어플리케이션이라 칭하고 오직 한 가지의 pointer 이벤트를 받는다고 하였다. 이러한 어플리케이션들은 자신의 Window 안에서만 rendering이 일어난다. 반대로 Multi-Touch 환경에 적합하게 만들어진 어플리케이션은 Novel 어플리케이션이라고 하였다. Novel 어플리케이션의 경우, Multi-Touch를 비롯한 다양한 input 이벤트를 받을 수 있다. 또 이 종류의 어플리케이션들은 전체 디스플레이 Window에서 rendering이 일어난다. Window Manager Server에 새로운 Multi-touch 관련 이벤트들을 등록한다. 이렇게 등록한 이벤트들은 Novel 어플리케이션에서 받아서 처리할 수 있도록 하고 Legecy 어플리케이션은 single-pointer event에 대한 처리만 할 수 있게 등록해주었다. 이러

한 방식으로 Multi-Touch 환경에서 다양한 input 양상에 대해 처리하는 방법을 제안하였다.

□ 다중 사용자/서비스 지원을 위한 멀티 스레드 기반 Window Manager 연구

대형 멀티터치 스크린의 대중화로 인해 테이블탑과 같은 새로운 환경들이 나오고 다중 사용자 환경 중심으로 변화하고 있는 가운데 기존에 이야기되지 않았던 Window Manager의 병렬화와 같은 최적화 기법의 필요성이 대두되었다. 이 논문에서는 그 필요성에 대응하여 기존에 단일 사용자 환경에서 싱글 스레드로 동작하는 Window Manager를 다중 사용자 환경에 맞게 멀티 스레드로 구현하여 제안하고 있다 [1].

현재의 Window Manager는 단일 사용자 환경에 맞게 구현되어 있기 때문에 부하가 증가하더라도 싱글 스레드를 유지한다. 그러나 다중 사용자 환경에서는 동시에 처리해야 하는 이벤트들이 크게 증가하게 되어 싱글 스레드 상태의 Window Manager는 병목이 될 가능성이 크다. 멀티 스레드 Window Manager는 윈도우 그룹 단위의 동기화 정책으로 구현하였다. 하나의 윈도우 그룹은 한 명의 사용자에 의해 실행되고 있을 가능성이 높기 때문에 윈도우 그룹 단위의 동기화 정책을 선택하였다. 입력 장치 관리의 경우, 멀티 터치에 대한 처리를 가장 고려하였고 멀티 터치가 여러 사용자에 의해 생성된 탭이므로 기본적으로 탭 간의 연관성이 없다고 봤다. 따라서 각각의 탭을 개별적으로 큐에 할당하였다. 또 이 논문에서는 RCU(READ-COPY-UPDATE)를 제안하고 있다. 윈도우 별로 RCU를 위한 큐를 가지고 화면이 갱신될 때마다 해당 큐에 새로운 버퍼를 추가한다. 컴포지터는 이러한 모든 윈도우의 큐들을 복사해 가져와 화면을 갱신하는 것이다. 이러한 방식은 READ-SIDE(컴포지터)에서는 WAITING-FREE를 보장하기 때문에 빠른 화면 갱신이 가능해진다.

이렇게 만든 멀티 스레드 Window Manager로 Wayland 프로토콜을 지원하게 하여 8개의 코어를 가지는 데스크탑 환경에서 5개의 윈도우 클라이언트가 각각 10000번 씩의 Wayland 프로토콜 요청을 보내 응답을 받는 시간을 측정하는 실험을 진행하였다. 싱글 스레드 Window Manager 환경에서는 20%의 CPU 사용률을 보였고 30초 정도의 시간이 걸렸다. 그러나 멀티 스레드 Window Manager 환경에서는 CPU 사용률이 100%에 가까이 측정되었고 총 4초의 시간이 걸렸다. 이 논문을 통해 다중 사용자 환경에서 Window Manager 최적화가 큰 성능상의 이점을 야기할 수 있다는 것을 알 수 있었다.

■ 제안 작품 소개

본 논문에서는 다중 사용자 환경에서 빈번하게 발생할 수 있는 윈도우(창) 크기 조절 / 종료 등의 Window Decoration 관련 이벤트들에 대해서 Window Decoration 방식이 서로 다른 Wayland(CSD와 X window(SSD)의 성능상의 차이를 비교하고자 한다.

실험 환경은 MacOS의 Parallels Desktop (가상환경 플랫폼)에서 Ubuntu Desktop 16.04 Virtual Machine을 설치하였다. CPU는 Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz 이고, 2 코어를 가지고 있고 메모리는 2GB이다. X window를 테스트하기 위해 X window System인 X11와 openbox 를 설치하고 Wayland 를 테스트하기 위해 Wayland compositor의 reference implementation인 Weston을 설치하였다.

벤치마크 툴로는 X11 server performance 테스트 프로그램인 x11perf로 벤치마크를 하였다. x11perf가 X11 server performance 테스트 프로그램이지만 Wayland는 X11 프로토콜에 맞게 만들어진 프로그램도 지원해야 하기 때문에 Wayland의 performance 테스트에도 사용할 수 있다. x11perf로는 벤치마크 테스트의 실행시간 밖에 알 수 없기 때문에 리눅스 성능 측정 도구인 perf로 벤치마크 테스트 실행 간의 메모리 및 CPU 사용량을 측정하였다.

벤치마크 테스트를 위한 x11perf의 옵션은 create, destroy, popup, move, resize, circulate를 사용하였다. create, destroy 옵션은 새로운 윈도우를 생성하고 윈도우를 종료시키는 작업을 하는 옵션이다. popup 옵션은 윈도우를 최소화를 하고 다시 불러오는 작업을 하는 옵션이다. move 옵션은 활성화된 윈도우를 움직이는 작업을, resize 옵션은 활성화된 윈도우의 크기를 조절하는 작업을 수행한다. circulate는 활성화되어있는 윈도우 중에 가장 밑에 있는 윈도우를 가장 위로 올리는 작업을 수행한다. 이러한 옵션들을 선택한 이유는 다중 사용자 환경에서 자주 발생할 가능성이 높은 Window Decoration 이벤트로 X window와 Wayland의 차이점인 Window Decoration 방식에 따른 성능 차이를 확인할 수 있기 때문이다.

■ 구현 및 결과분석

```
if test -f result.csv; then
    rm result.csv
fi

options=( "create" "destroy" "popup" "move" "resize" "circulate" )
echo "OPTIONS,TASK_CLOCK,CONTEXT_SWITCHES,PAGE_FAULT,TIME" >> result.csv

for i in "${options[@]}"
do
    sudo perf stat x11perf -$i 2> tmp.txt
    TC=$(cat tmp.txt | grep task-clock | awk '{ print $1 }' | sed "s/,//")
    CS=$(cat tmp.txt | grep context-switches | awk '{ print $1 }' | sed "s/,//")
    PF=$(cat tmp.txt | grep page-fault | awk '{ print $1 }' | sed "s/,//")
    TIME=$(cat tmp.txt | grep seconds | awk '{ print $1 }' | sed "s/,//")
    echo $i,$TC,$CS,$PF,$TIME >> result.csv
done
```

그림 4 x11perf 벤치마크 테스트 쉘 스크립트

그림 4는 x11perf를 이용해 벤치마크 테스트를 수행하고 결과를 csv 파일로 저장하는 쉘 스크립트 코드이다. options에 벤치마크 테스트를 수행할 x11perf 옵션들을 작성해주고 그에 따라 for문으로 반복하게 하였다. x11perf를 실행하는 동안 perf를 통해 Task Clock, Context Switches, Page Fault, Time 등을 측정하여 tmp.txt 파일에 저장하였다. 저장한 내용에서 결과 분석에 필요한 부분들을 grep과 awk, sed 명령어를 사용해 result.csv 파일에 저장하였다. csv 파일로 저장하여 결과 분석이 용이하도록 하였다.

이번 보고서에서는 다중 사용자 환경에서 여러 사용자로 벤치마크 테스트를 하는 것이 아닌 단일 사용자 환경에서 하나의 사용자가 벤치마크 테스트를 수행하는 결과를 분석하고자 한다. 아래 표 1과 표 2는 각각 X window 환경 (X11 + openbox)에서 실행한 결과와 Wayland 환경 (weston)에서 실행한 결과이다. 각각 create, destroy, popup, move, resize, circulate 옵션으로 Task Clock, Context Switches, Page Fault, Time을 측정하였다.

표 1 X Window

OPTIONS	TASK_CLOCK	CONTEXT_SWITCHES	PAGE_FAULT	TIME
create	1927.195033	264361	187	5.587053225
destroy	1725.562128	176963	187	4.970816729
popup	2015.375438	233894	183	234.9226288
move	2784.298388	236816	183	250.9397907
resize	2544.80609	233755	183	228.888161
circulate	3481.008685	518722	183	244.4045789

표 2 Wayland

OPTIONS	TASK_CLOCK	CONTEXT_SWITCHES	PAGE_FAULT	TIME
create	1845.157433	169725	187	5.418965188
destroy	1370.317327	118517	187	4.105518135
popup	2038.877535	192198	181	245.4850232
move	1851.670217	157360	184	255.4090834
resize	2225.433342	146504	181	240.0235858
circulate	1778.68737	208623	181	234.9133719

그림 5 ~ 8는 표 1과 표 2를 가지고 같은 벤치마크 테스트 옵션으로 실험했을 때의 X window와 Wayland의 성능 차이를 그래프로 표현한 것이다. 각각의 벤치마크 테스트 옵션에서 Wayland와 X Window의 Task Clock을 비교한 것에서는 popup 옵션을 제외한 다른 테스트들에서는 모두 Wayland가 X Window 보다 좋은 성능을 보여주었다. Context Switch 횟수는 모든 옵션에서 Wayland가 X Window 보다 좋은 성능을 보여주었다. Page Fault 횟수는 Wayland와 X Window가 1 ~ 3 정도의

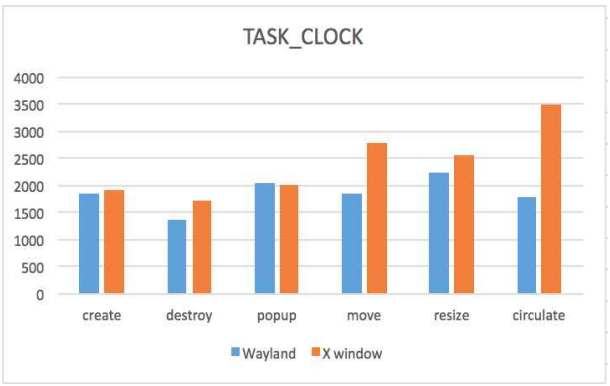

그림 5 Task Clock 성능 비교

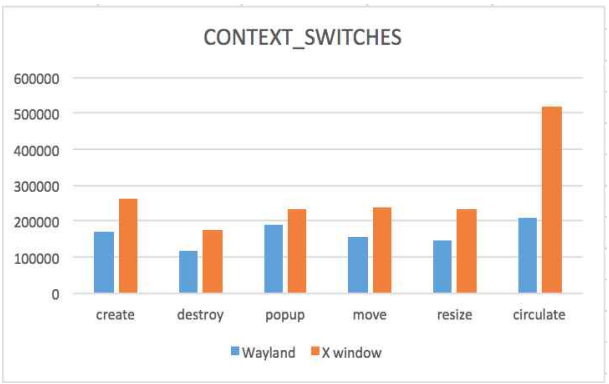

그림 6 Context Switches 성능 비교

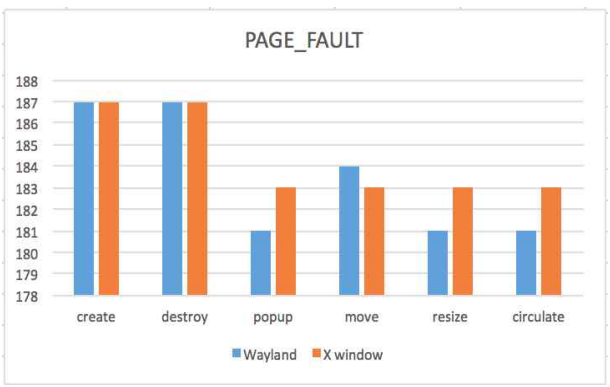

그림 7 Page Fault 성능 비교

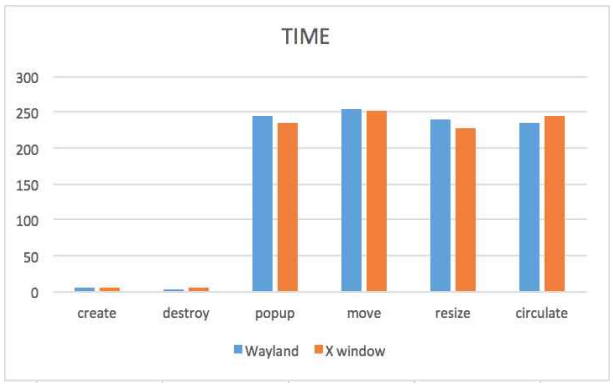

그림 8 Time 성능 비교

차이밖에 보이지 않아 거의 비슷한 수준을 보여주었다. Time의 경우는 create, destroy, circulate 옵션에서는 Wayland가 더 적은 시간을 소모하였고 나머지 popup, move, resize 옵션에서는 X Window가 더 적은 시간을 소모하였다.

다중 사용자 환경에서 실험한 것이 아니라 단일 사용자 환경에서 실험했지만 분석 결과 대부분의 경우에서 Wayland가 좋은 성능을 보였지만 가장 중요한 Time 부분에서 X Window가 좋은 성능을 보여주었다. 이 후의 연구에서는 다중 사용자 환경에서의 실험을 진행하고 단일 사용자 환경에서 실험했던 결과와 비교 / 분석해볼 예정이다.

■ 결론 및 소감

본 논문을 통해 Window Decoration 방식이 다른 Window Manager에 따른 Window Decoration 이벤트 처리 시의 성능 차이를 분석하였다. 다중 사용자 환경에서 어떤 Window Decoration 방식을 선택해야 사용자에게 더 좋은 성능 상의 이점을 가져다 줄 수 있는지 분석을 진행하였다.

본 논문 및 연구를 진행하면서 대학교 4년 동안 학부 수업에서는 해볼 수 없는 경험들을 많이 했다. 외국 논문 / 국내 논문 가릴 것 없이 여러 논문들을 읽었고 리눅스 커널에 대해서도 깊이 있는 공부를 할 수 있었다. 리눅스라는 운영체제를 깊게 공부한 것이 이번이 처음이었고 어려움이 많았다. 공부를 진행하는 과정에서 모르는 것이 있거나 이해가 안 되는 것이 있을 때, 석박사 과정 분들이 많은 도움을 주셨다. 처음 논문을 작성해보는 것이기 때문에 전체적인 프로세스를 몰랐지만 이 과정에서도 석박사 과정 분들에게 도움을 많이 받았다.

본 보고서에서는 X Window와 Wayland의 Window Decoration을 비교하여 분석하였다. 최종 보고서까지는 어떠한 Window Decoration이 어떤 특징 때문에 이런 성능에서는 좋은 결과를 보였는지 분석하고 작성할 예정이다. 본 논문에서는 이미 존재하는 Window Decoration 방식들에 대한 분석 및 비교로 이루어진 수준이지만, 이 후 더 공부한다면 새로운 Window Decoration 방식을 제안하거나 성능 향상을 위한 수정을 제안할 수 있을 것이다.

■ 참고문헌

[1] 김인혁, 김정한, 김태형, 엄영익, "다중 사용자/서비스 지원을 위한 멀티 스레드 기반 윈도우 매니저 연구," *한국정보과학회 한국컴퓨터종합학술대회 논문집*, pp. 1291-1293, 2014.

[2] 홍다훈, 이민훈, 엄영익, "다중 사용자 환경에서 응답성 향상을 위한 CFQS 스케줄러 구현," 한국정보과학회 학술발표논문집, pp. 1768-1770, 2016.

[3] X.org [Online]. Available: https://www.x.org/wiki (downloaded 2016, Nov. 10).

[4] Wayland. [Online]. Available: https://wayland.freedesktop.org/ (downloaded 2017, Mar. 10).

[5] H. S. Kim, S. J. Noh, and Y. I. Eom, "Comparative Analysis of the Performance of X Window and Wayland Systems," *Proc. of the KIISE Winter Conference,* pp. 1791-1793, 2016.

[6] R. Wimmer and F. Hennecke, "Everything is a Window: Utilizing the Window Manager for Multi-Touch Interaction," *Proc. of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems,* pp. 1-4, 2010.

[7] M. Lee, I. Kim, and Y. I. Eom, "Analyses of I/O Schedulers for Multi-user Systems Considering Periodicity of Interactive Processes," *Proc. of the KIISE Winter Conference*, pp. 1278-1280, 2015.

[8] I. Leftheriotis, K. Chorianopoulos, and L. Jaccheri, "Design and Implement Chords and Personal Windows for Multi-user Collaboration on a Large Multi-touch Vertical Display," *Journal of the Human-centric Computing and Information Sciences,* Vol. 6, No. 14, pp. 1-19, 2016.

[9] Sadia, H. and Zin, A. M, "Feature Identification of an Interactive Multiuser Meeting Tabletop based on Intuitive Gesture Recognition," *Journal of Theoretical and Applied Information Technology,* Vol. 92, No. 2, pp. 354-364, 2016.