**<PROJECT 기말보고서>**

**WebOS 오픈소스**

**ABI 변경 관리 툴 개발**

**팀원: 전영웅, 이희재**

**교수님: 신영길**

**회사 담당자: 송효진 (LG 전자)**

**Table of Contents**

**1.** **Abstract**

**2.** **Introduction**

**3.** **Background Study**

**3.1 관련 접근방법/기술 장단점 분석**

**3.1.1 WebOS**

**3.1.2 ABI Compatibility**

**3.1.3 ABI Compliance Checker**

**3.2 프로젝트 개발환경**

**4.** **Goal/Problem & Requirements**

**5.** **Approach**

**6.** **Project Architecture**

**6.1 Architecture Diagram**

**6.2 Architecture Description**

**7.** **Implementation Spec**

**8.** **Specifications Changes**

**9.** **Future Work**

**10.** **Division & Assignment of Work**

**11.** **Schedule**

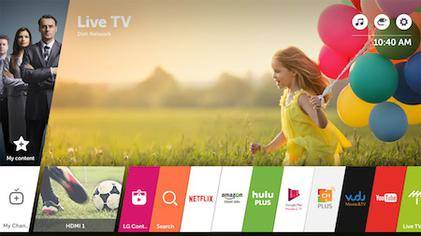
**[Reference]**

1. **Abstract**

  IoT 시대, 일상에서 사용하는 백색 가전과 각종 기기들이 인터넷을 가지고 ‘스마트'해지고 있다. 스마트폰 시장에서는 안드로이드와 iOS가 과점하는 형태로 굳어졌지만, IoT 장치에서는 글로벌 유명기업마다 각자의 OS를 운용하고 있다. LG 전자는 냉장고, 스마트 TV, 웨어러블, 시나이저 등에서 WebOS를 플랫폼으로 선택하여 사용하고 있다. 앞으로의 IoT 시장 상황에서, WebOS는 확장성, 안정성, 개발 편의성이 검증된, 적합한 플랫폼으로 평가 받고 있다. 이러한 WebOS 개발에 도움을 주고자 하는 목적으로 이번 프로젝트가 기획되었다. WebOS의 규모가 작지 않은 만큼, 업데이트 상황에서 발생할 수 있는 backward compatibility 문제를 효율적으로 관리할 필요가 있다. 이를 돕기 위해, ABI 변경과 관련하여 발생할 수 있는 문제를 업데이트 발생 시 자동으로 찾아주고, 개발자에게 직관적이고 시각적으로 정리하여 정보를 제공하는, 웹 서비스를 제작하고자 하였다.

**2. Introduction**

LG 전자는 HP로 부터 WebOS를 인수한 이후로, 제품들에 WebOS를 적극 활용하고자 노력하고 있다. 특히, 스마트 TV의 경우 기존 WebOS가 가진 feature인 Card View를 활용하여 기존 TV에 비해 사용성을 크게 개선한 것으로 평가 받고 있는 만큼, WebOS에 대한 기대가 작지 않다고 할 수 있다. IoT 시대에서 플랫폼의 역할은, 이러한 사용성 개선 사례에서 알 수 있듯이, 서비스의 질적인 측면에서 가장 중요하다고 할 수 있다. 또한, 플랫폼의 효율성은 전체 IoT device의 효율성과도 직접 연관되며, 플랫폼이 가지는 능력에 따라 앞으로 할 수 있는 일이 제한되기도, 가능하기도 할 것이다. 따라서 빠르고 경제적인 플랫폼을 만들기 위해서 지속적인 노력은 필수적이다.

(WebOS 스마트 TV)

WebOS가 이러한 성공적인 플랫폼으로 성장하기 위해서, 오픈소스 업데이트마다 헤더, so 파일 변경에서 발생할 수 있는 backward compatibility(ABI compatibility) 문제를 효과적으로 관리하고자 하였다. WebOS와 같이 규모가 큰 오픈소스에서는 수 많은 shared library를 참조하고 있으며, 또한 수 많은 오픈소스 라이브러리는 저마다의 개발자, 관리자에 의해 독자적으로 업데이트 및 관리된다. 따라서 규모가 큰 만큼 ABI compatibility에 대한 위협도 크다고 할 수 있다. 어떤 라이브러리 업데이트 후 ABI compatibility가 깨지게 되면, 라이브러리를 사용하는 모듈로 하여금 재컴파일을 필요로 하게 만들거나 run time error를 발생시킨다. 의도하지 않은 문제 발생으로 인한 피해를 막기 위해, 문제 발생 시 개발자가 변경한 버전에서 문제를 찾아내고 알려주어 관리를 돕는 툴을 구현하고자 한다.

WebOS의 ABI 변경 관리를 위한 툴 개발을 프로젝트의 목표로 설정하여 진행하였다. 중간 발표 이전까지 수행한 것은 다음과 같다. 우선적으로 WebOS의 기본 구조를 이해하고, 버전 변경이 일어날 때 어떠한 변화가 일어나는지 알기 위해서 WebOS 자체를 이해하고 직접 빌드해보고자 하였다. 이후 WebOS의 서로 다른 두 버전을 활용하여, ABI incompatibility에 대한 이해를 높이고자 하였다. 이를 위해, ABI incompatibility를 발견하는 도구인 ABI Compliance Checker를 사용하였다. Checker의 input을 적절하게 설정하고, 수행 결과를 확인하였다. 결과를 분석하고 이후 프로젝트 수행에서 결과 활용 방안에 대해서 논의하였다.

중간 발표 이후, 개념들에 대한 이해를 바탕으로 효과적인 ABI 변경 관리를 위한 인프라 및 정보 열람을 위한 웹 앱을 구현하였다. 툴에 속한 모듈들은 WebOS에서 버전 업데이트 여부를 지속 탐지하며, 업데이트 발생 시 자동으로 이전 버전과 업데이트된 버전을 이용하여 ABI Compliance Checker 도구를 수행, 결과를 만들어낸다. 생성된 리포트를 웹 페이에서 조회할 수 있게끔 만들며, 리포트 내용을 간단하게 요약한 것을 회사 내부 커뮤니케이션 시스템인 JIRA를 통해서도 제공해주는 것을 전체 서비스로 계획하였다.

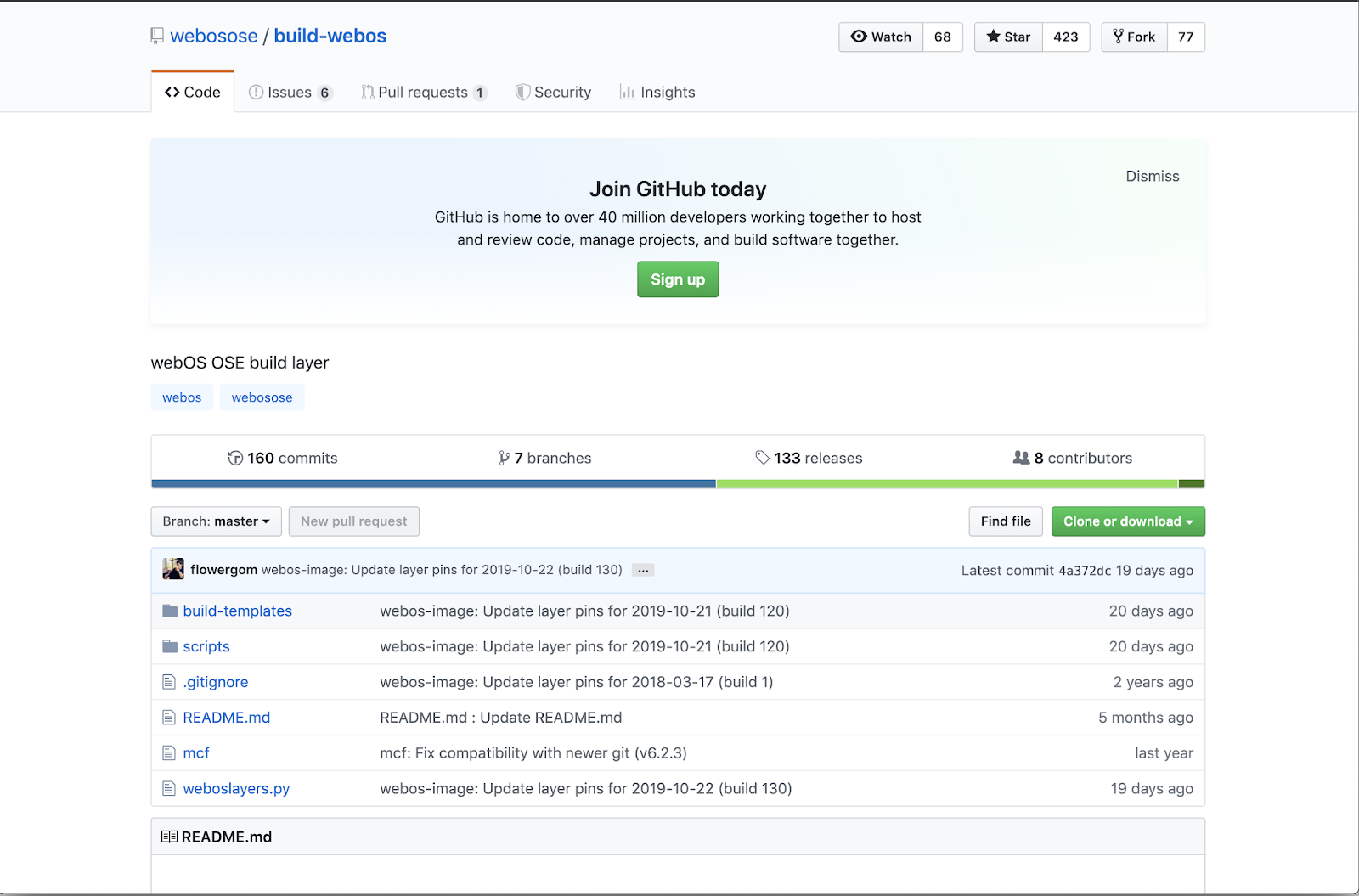
보고서에서는 프로젝트에 필요한 개념을 이해하기 위한 Background Study에 대해 다룬 뒤, 개념들을 활용하여 문제와 필요조건에 대해서 서술하였다. 이후 전체적인 architecture를 간단히 정리하고, 프로젝트 수행 과정을 어떻게 설계하였는지 설명한다. 이후 기말 발표까지 어떠한 프로젝트 수행 과정을 거쳤는지 상세한 내용을 이야기할 것이다.

**3. Background Study**

**3.1    관련 개념 및 접근 방법**

3.1.1 WebOS

  LG는 HP로 부터 WebOS를 인수하여, 스마트 TV, 가전 제품, 웨어러블용으로 계획 및 활용하고 있다. 2018년 3월 open source edition으로 공개하였으며, 사이트를 통해 개발 가이드, 포럼 등을 운영하고 있다.



  WebOS 깃헙 페이지를 통해 플랫폼 코드를 내려받아 빌드를 진행하였고, 새로운 버전 release가 올라오게 되면 새로운 버전을 내려받아 빌드하였다. WebOS의 빌드 시스템은 Yocto Project를 이용하고 있는데, 이는 규모가 큰 linux 기반 시스템을 빌드하기에 적합하다. 이중 bitbake 파일이 실제 빌드를 담당하여 빌드하고자 하는 source를 download하고 최종 install까지의 과정 진행한다. 빌드 시 recipe라고 불리는 .bb 파일들을 읽게되고 여기에 정의된 실행 단위인 Task에 따라서 빌드를 진행한다.

3.1.2 ABI compatibility(Backward compatibility)

  ABI는 API와는 구별되는 개념으로 기계 수준, 이진값 수준에서의 인터페이스를 이야기하는 것으로, API가 소스코드 레벨에서의 호환을 다룬다면 ABI는 기계 수준에서의 호환을 다룬다. 쉽게 말해, 같은 바이너리 값을 서로 다른 두 기계에서 실행하였을 때 같은 결과를 만들어낸다면 기계 수준에서의 호환이 이루어진다고 할 수 있다. ABI는 기계어 코드가 사용하는 CPU 명령어 집합, 메모리 사용 방식, 실행 가능한 바이너리 형식(프로그램, 공유 라이브러리) 등의 정보를 포함한다. armeabi, x86이 대표적인 ABI라고 할 수 있다..

  linux에서 컴파일한 실행파일이 window에서 실행하였을 때 같은 결과를 내지 않는 경우가 존재하는데, 두 가지 이유가 있을 수 있다. 운영체제에서 사용하고 있는 바이너리 명령어 집합이 다르거나 공유 라이브러리가 다르기 때문이다. 이러한 경우 ABI compatibility가 깨졌다고 이야기하며 다음과 같이 해결할 수 있다. 전자의 경우 예를 들어 x86에서 사용되도록 만들어진 프로그램을 arm에서 사용하고자 한다면 arm에서 사용되는 컴파일러를 이용하여 새로 컴파일 하는 방법으로 위의 문제를 해결할 수 있다. 후자의 경우, 구버전과 신버전 라이브러리간 호환이 안되는 경우라고 볼 수 있는데 함수의 내용이 바뀌거나, export된 data가 바뀌거나 export 함수가 제거, 인터페이스가 변하는 경우 이러한 일이 발생할 수 있다.

  ABI incompatibility가 발생하는 경우 구버전 라이브러리에서는 정상 작동하던 앱이 신버전 라이브러리로는 앱이 실행되지 않는다던가, 예상하는 결과와 다른 결과를 만들어낸다던가 하는 결과를 만들어낸다. 따라서 개발자들은 이러한 incompatibility 발생을 의도하지 않은 경우, incompatibility의 존재를 신속하게 파악하여 문제를 해결해야 한다. 경우에 따라 신버전의 라이브러리가 compatibility를 유지할 수 있도록 하고, 의도한 경우 재컴파일의 필요를 알려야 한다..

3.1.3 ABI Compliance Checker(ABICC)

  구 버전 라이브러리와 신 버전 라이브러리를 input으로 하여 ABI compatibility를 체크해주는 툴이다. 프로젝트는 WebOS 개발자들이 업데이트로 인해 예상치 못한 ABI incompatibility를 겪지 않도록 이 도구의 실행 결과를 잘 정리하여 웹 서비스 형태로 정보를 제공해주는 것을 목표로 하고 있다. ABICC는 오픈소스 형태로 배포되어 있으며 다음과 같은 원인의 compatibility 문제들을 찾아낼 수 있다.

* Problems with Data types: structure에서 필드 타입, 순서, 변경, 사이즈 변경 등, class에서 가상 함수 변경, 가상 함수 overwrite 등.
* Problems with Symbols: symbol의 삭제, attribute 변경, parameter 이름 변경 등
* Problems with Constants(#define): changed value

**3.2    프로젝트 개발환경**

   WebOS 빌드 및 ABI Compliance Checker 도구가 요구하는 환경으로 Linux를 선택하여 운용하였고, 이후 웹 서비스 개발을 위해서 apache, HTML, CSS, JS, PHP를 이용하였다. 서버 개발을 위해 AWS의 EC2 서비스를 사용하였고, 서버의 동작을 자동화 하기 위해 쉘 스크립트(bash script)를 이용하였다.

**4. Goal/Problem**

   WebOS가 참조하는 다양한 오픈소스들의 독자적인 버전 업데이트로 인한 ABI 변경에 대해 효과적인 관리에 대한 필요성이 존재한다. ABI Compatibility는 개발자/사용자 입장에서 관리될 필요가 있는 사항이다. 따라서 WebOS 개발자에게 ABI compatibility와 관련한 정보를 시각적, 효과적으로 제공하는 웹 서비스를 구현하여, 개발자들을 지원하고자 한다. 프로젝트를 수행하는 학생들은 큰 규모의 오픈소스를 다루는 과정에서 라이브러리 간의 dependency를 파악하고, 플랫폼 버전 업데이트에서 벌어지는 일들을 접하며 오픈소스 구조에 대해서 이해도를 높이고 간단한 웹 서비스 구현 개발 경험을 얻게 될 것이다.

**5. Approach**

우선은 WebOS를 처음 접하였기 때문에, WebOS에 대한 이해를 높이기 위해 직접 빌드해보는 시간을 가졌다. 내부 라이브러리 사이의 관계를 파악하고, 플랫폼 버전이 달라진다는 것이 어떤 의미를 가지는지 파악할 수 있을 것으로 기대했다. 두 버전의 WebOS 빌드를 확보한 뒤에는 ABI Compliance Checker(이하 ABICC)를 사용하여 ABI compatibility에서 어떠한 문제가 발생하는지, 어떠한 경우에 어떤 문제를 결과물로 내놓는지를 알고자하였다. 원리와 수행 과정 및 결과에 대한 이해를 거쳐, 프로젝트에서 직접 다뤄야 하는 헤더와 so를 추가하고, 웹 페이지에 포함시켜야 할 내용을 구상하였다.

  최초에는 ABICC로 WebOS가 참조하는 모든 헤더와 so 파일에 대해 compatibility를 검사하고자 하였다. 하지만, 회사 측과 상의 결과 LG 전자에서 주로 변경을 가하는 부분에서 우선적으로 compatibility를 검사하는 것이 중요하다고 판단하였고, 이에 헤더와 so 파일 일부를 선별하여 계속 진행하였다. 또한 ABICC가 내놓는 결과물의 질이 충분하다고 판단, 따로 커스터마이징, 변경은 가하지 않았다. 따라서 만들어진 보고서를 자동으로 업로드하여 조회할 수 있는 간단한 웹 페이지를 구현한 후 JIRA 서버에 보고서의 핵심 부분을 요약한 내용과 함께 이슈를 만들었다.

  요약된 프로젝트 수행 단계는 다음과 같다.

1. 특정 두 버전의 WebOS 코드를 가져오기 위해서, git log 명령어를 통해 #build가 명시된 커밋의 해시태그를 활용하고 git checkout 명령어를 실행한다.

2. WebOS OSE 오픈소스를 다운로드 받아서 가이드에 따라 빌드한다. (리눅스 환경 필수)

3. 두 버전의 빌드 결과를 확보하여, ABICC 도구를 수행한다.

4. 도구 사용 시, bb 파일에 명시된 라이브러리 간 의존성 정보를 참고하여, webOS 오픈소스의 파일 구조에 맞게 적절한 실행 옵션을 검토한다.

 중간 이후 수행 요약

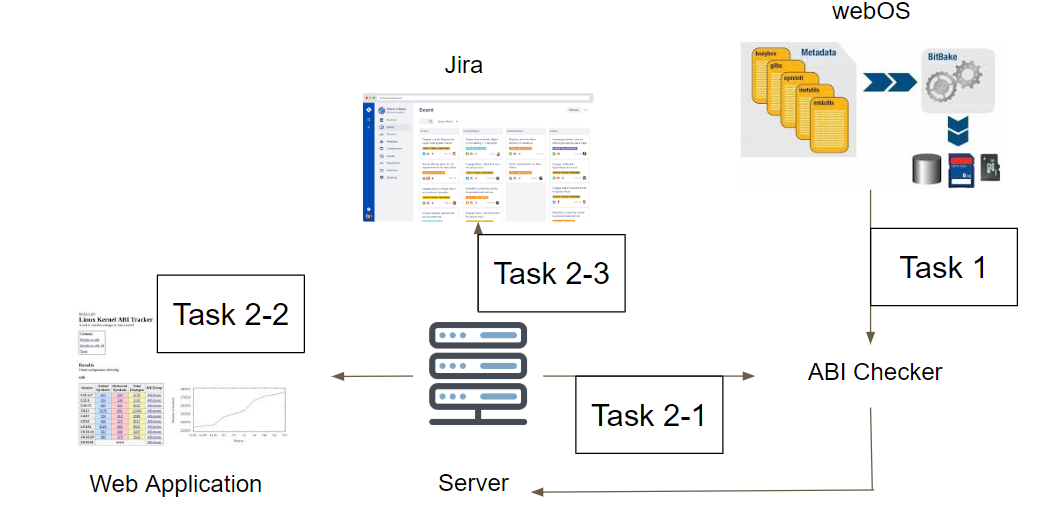
  5. 자동화를 위해 WebOS github 페이지에서 릴리스가 발생하면 새로운 버전의 WebOS를 서버에 받고, 빌드를 진행하였다.

  6. 빌드를 성공하면, 선별한 헤더와 so 파일을 input으로 하기 위한 xml 파일을 생성한다. 이를 구 버전 WebOS 빌드의 xml과 함께 ABICC를 수행하여 결과물을 만들어낸다.

  7. ABICC의 결과물, 리포트를 웹 페이지에서 조회할 수 있게끔 하고, JIRA 서버에 요약된 내용을 이슈 형태로 제공한다.

**6. Project Architecture**

**6.1 Architecture Diagram**



**6.2 Architecture Description**

Webos의 github에서 새로운 release가 생기면 서버에서 자동으로 다운받아 빌드를 시작한다. 빌드가 완료되면 예전 webos 빌드파일과 abi-compliance checker를 돌린다. 돌린 결과는 webos 개발자가 볼 수 있도록 웹으로 올라가게 된다. 또한 결과는 자동으로 LG 전자 개발팀의 Jira에 새로운 이슈로 등록이 된다. 이 모든 프로세스는 자동으로 돌아가게 만들어진다. 유저는 웹에 접속해서 어떤 abi가 달라졌는지 확인할 수 있다. 모든 동작은 자동으로 돌아가기 때문에 서버는 항상 돌아가야 하고 유저와의 interaction은 실질적으로 많지가 않다. 그렇기 때문에 벡엔드를 얼마나 잘 구현하느냐가 중요하다.

**7. Implementation Spec**

**7.1 Task 1 AUTOMATIC BUILD**

Github에 새로운 release를 받기 위해서는 abi-monitor라는 툴을 사용했다. 이 툴은 자동으로 github에 새로운 release가 나왔는지 확인 하고 다운로드 해주는 툴이다. 하지만 만약 이미 release가 서버내에 존재한다면 새로운 release가 없는 것으로 판단하고 다운로드를 하지 않는다.

만약 새로운 release 버전이 발견이 되었다면 가장 오래된 빌드 파일은 삭제된다. 그 이유는 빌드하기 위해 100GB 이상의 메모리가 필요한데 abi-checker를 돌리기 위해서는 두가지 버전만 필요하기 때문에 안 쓰이는 버전은 지우는게 좋다고 판단했기 때문이다. 또한, 새로운 빌드파일이 생길경우 자동으로 빌드를 해주는 script를 만들었다. 이 script내에는 빌드 파일의 pathname을 저장하고 webos 빌드를 실행을 시키는 코드 몇줄이 적혀져 있다.

다음은 이 프로세스를 코드로 변환한 script 코드이다.

echo "get webos new release"

abi-monitor -get webos.json

newVersion=$(ls -td ./src/webos/\*/ | head -1)

newVersion="$(basename ${newVersion})"

if [ "$oldVersion" != "$newVersion" ]; then

echo "new version found!"

echo "remove old build file"

sudo rm -rf "$(ls -trd ./installed/webos/\*/ | head -1)"

echo "extract new release"

dir="./src/webos/$newVersion"

mkdir ./installed/webos/$newVersion

for entry in "$dir"/\*

do

echo "$entry"

tar zxvf "$entry" -C ./installed/webos/$newVersion

done

echo "build new release starting..."

buildfileName=$(ls -d ./installed/webos/$newVersion/\*/|head -n 1)

echo "$buildfileName"

cp ./build\_scripts/webos.sh $buildfileName

cd $buildfileName

./webos.sh

webos가 빌드가 되고 나면 이제 abi-compliance-checker를 돌리기 위해서 몇가지 준비를 시작한다. 우선 새로운 OLD.xml과 NEW.xml안에 path를 새롭게 바뀐 빌드파일의 path로 바꿔줘야 한다. 이를 위해서 linux안에 있는 sed를 이용했다.

다음은 sed를 이용해 xml 파일을 수정하는 코드이다. 파일의 line number를 이용했다.

sed -i "2s%.\*%$newVersion%g" NEW.xml

sed -i "6s%.\*%$wayland%g" NEW.xml

sed -i "7s%.\*%$pixman%g" NEW.xml

sed -i "8s%.\*%$icu%g" NEW.xml

sed -i "9s%.\*%$libxkbcommon%g" NEW.xml

sed -i "10s%.\*%$libexif%g" NEW.xml

sed -i "11s%.\*%$dbus%g" NEW.xml

sed -i "12s%.\*%$libcap%g" NEW.xml

sed -i "13s%.\*%$elfutils%g" NEW.xml

sed -i "14s%.\*%$nspr%g" NEW.xml

sed -i "15s%.\*%$curl%g" NEW.xml

sed -i "16s%.\*%$waylandEx%g" NEW.xml

sed -i "20s%.\*%$newPath%g" NEW.xml

**7.2 Task 2-1 ABICC**

Abi-Compliance-checker를 돌리기 위한 준비는 다 끝나면 이제 abi-checker를 돌리면 된다. 하지만 새로운 checker가 오류를 낼 경우를 대비해 실패시 이메일을 보내도록 설정을 해놨다. checker가 완료가 될 경우 /var/www/html 로 결과 파일을 옮긴다. 여기서 옮겨진 파일은 자동으로 index.php가 웹에 올린다. 만일 이미 결과 파일이 웹에 올라가 있을 경우 옮기는 작업은 중단되도록 해놨다.

echo "run abi-compliance checker"

abi-compliance-checker -gcc-options -Werror -lib webos -old OLD.xml -new NEW.xml

if [ "$?" = 1 ]

then

echo "success"

echo "move reports to root html directories"

tmp=$(ls -td ./compat\_reports/webos/\*/ | head -n 1)

tmpname="$(basename $tmp)"

reportName=$tmpname"\_report.html"

cd /var/www/html

if [ ! -f "$reportName" ]; then

cd ~

sudo cp $tmp"compat\_report.html" $reportPath$reportName

#create Jiraissue

sudo ./jiraissue/createIssue.sh $reportPath $reportName

fi

exit 2

else

echo "fail"

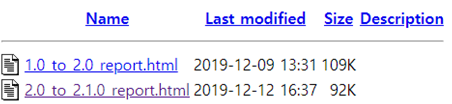
echo "sending mail..."

echo "failed to run abi checker" | mail -s "ABI failure report" "#email"

fi

**7.3 Task 2-2 Web Page**

서버내에 Apache를 설치하고 index.php를 메인 웹페이지로 설정을 해뒀다. Amazon cloud server에서 고정 ip를 할당받아 52.39.65.72로 들어가면 우리 웹 사이트가 뜨도록 설정을 했다. index.php내에서는 서버에 있는 모든 결과 파일들을 생성된 날짜 순서대로 웹 페이지에 로드를 한다.



date\_default\_timezone\_set("Asia/Seoul");

// Reports Directory

$dir = "/var/www/html";

$reportname = "report.html";

// Open directory

$od = opendir($dir);

// Reports in the dir

$files = array();

while(($filename = readdir($od))) {

if($filename == "." || $filename == "..") {

continue;

}

// Reports are html files.

if(is\_file($dir . "/" . $filename) && strpos($filename, $reportname)) {

$files[] = $filename;

}

}

closedir($od);

rsort($files);

**7.4 Task 2-3 Jira Issue**

보고서 생성 이후, JIRA issue 생성을 위한 동작은 다음과 같다. JIRA에서 제공하는 API를 이용, 해당 jira 프로젝트의 서버 URI와 프로젝트 식별키, 프로젝트 관리자 아이디, 보안을 위한 API token을 config.json 파일로 입력받아 진행한다.

reportName=$2

newReportName=$1$2

projectUri=$(cat ~/jiraissue/config.json | jq -r .jirauri)

projectKey=$(cat ~/jiraissue/config.json | jq -r .projectkey)

jiraUsername=$(cat ~/jiraissue/config.json | jq -r .email)

jiraApiToken=$(cat ~/jiraissue/config.json | jq -r .apitoken)

또한 issue의 content 구성을 위해서는 ABICC 리포트를 파싱하여 적절한 text를 만들어낸다. 이를 위해 html 파싱을 지원하는 xmllint를 이용하였다.

binCompatibility=$(xmllint —noout —html —xpath $binCompatibilityPath'/text()' $newReportName 2>xmllint.error)

binRemovedSym=$(xmllint —html —xpath $binProblemPath'/tr[3]/td/a/text()' $newReportName 2>xmllint.error)

binProblemDTHigh=$(xmllint —html —xpath $binProblemPath'/tr[4]/td/a/text()' $newReportName 2>xmllint.error)

binProblemDTMedium=$(xmllint —html —xpath $binProblemPath'/tr[5]/td/a/text()' $newReportName 2>xmllint.error)

binProblemDTLow=$(xmllint —html —xpath $binProblemPath'/tr[6]/td/a/text()' $newReportName 2>xmllint.error)

binProblemSymHigh=$(xmllint —html —xpath $binProblemPath'/tr[7]/td/a/text()' $newReportName 2>xmllint.error)

binProblemSymMedium=$(xmllint —html —xpath $binProblemPath'/tr[8]/td/a/text()' $newReportName 2>xmllint.error)

binProblemSymLow=$(xmllint —html —xpath $binProblemPath'/tr[9]/td/a/text()' $newReportName 2>xmllint.error)

API를 위한 input들을 적절하게 가공한 뒤, issue.json 파일에 issue attribute 값들을 저장한다.

issueFields="{\"fields\": {\"project\": {\"key\": \"$projectKey\"}, \"summary\": \"$reportName is created.\", \"description\": \"$description\", \"issuetype\": {\"name\": \"Bug\"}, \"priority\": {\"name\": \"High\"}}}"

#\"timetracking\": {\"originalEstimate\": \"0m\", \"remainingEstimate\": \"1w\"}

curlData=$(echo $issueFields > issue.json)

마지막으로 curl 명령을 통해 JIRA issue를 생성하였다.

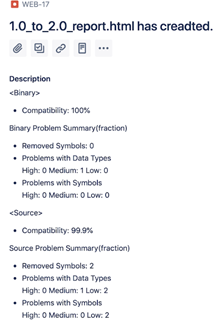
curl -D- -u $jiraUsername":"$jiraApiToken -X POST --data @issue.json -H """Content-Type: application/json""" https://$projectUri"/rest/api/2/issue/"

다음과 같이 결과가 정상적으로 생성됨을 확인할 수 있다.

<Jira issue List>



<Jira issue detail> Summary 내용을 통해 Problem Severity가 Medium 이상인 부분의 개수를 확인할 수 있다.



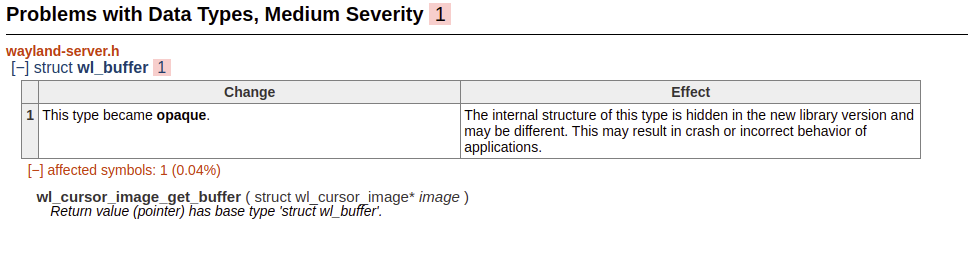
**7.5 Task 2-4 ABICC ON/OFF**

원래 구현할 목표에는 없었지만 웹 페이지에서 이 모든 프로세스를 끌 수 있는 기능을 넣기로 했다. 단순히 유저가 버튼을 누르면 crontab을 끄는 명령어를 실행하도록 설정하려 했지만 이는 sudo 명령어를 요구를 했다. 일반 유저가 sudo 명령어를 실행하기에는 많은 보안 문제가 있을거라고 예상해 단순히 active / inactive라는 폴더를 생성하게 만들었다. 그래서 스크립트 파일에서 inactive 폴더가 있으면 abi-checker를 멈추게 된다.

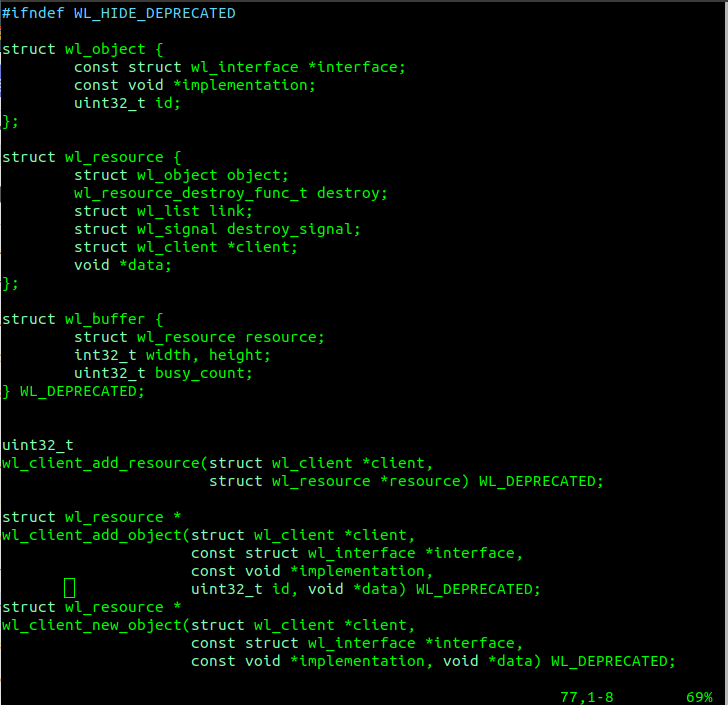
**Results**

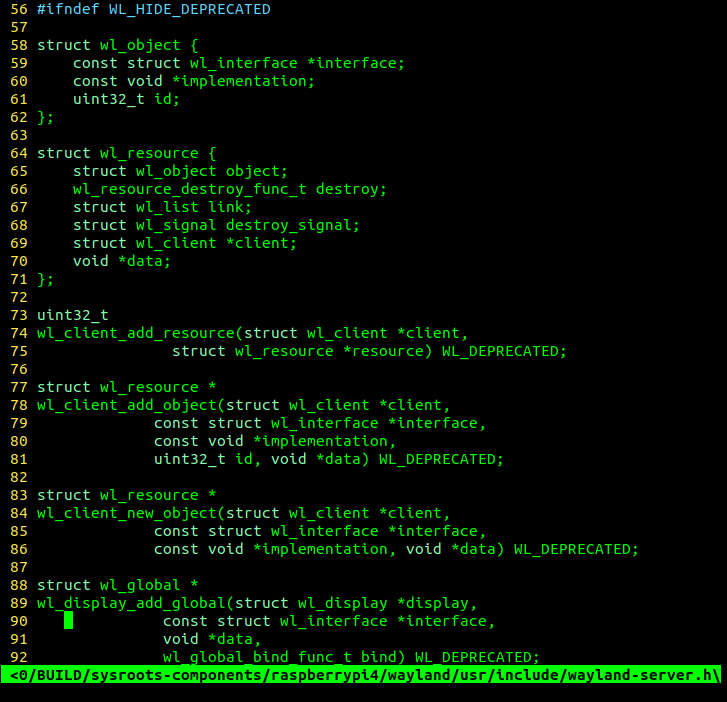
다음은 실제로 ABICC가 제대로 비교를 했는지 알아보기 위해 코드를 직접 비교를 해보았다.

실제로 Webos Version 1.0 과 Webos 2.0 ABICC 리포트에서 다음과 같은 변화가 있었다고 보고했다.

****

실제로 이런 변화가 있었는지 알아보기 위해 1.0과 2.0의 wayland-server.h를 비교해 보았다.



**Figure 1. Webos 1.0 wayland-server.h  
**

**Figure 2. Webos 2.0 Wayland-server.h**

\*코드 상에서 **struct wl\_buffer**가 사라진 것을 볼 수 가 있다. 때문에 ABICC에서는 문제가 생길 수 있다고 보고 했다.

**8. Specifications changes**

1. ABICC 리포트가 충분히 보기 좋게 결과물을 보여준다고 판단하여 다시 시각화할 필요가 없다고 판단했다.
2. 원래 계획상 Webos가 쓰는 모든 헤더파일에 관하여 ABICC를 돌리려고 했다. 하지만 ABICC같은 경우 컴파일 된 파일을 비교하기 때문에 컴파일 에러가 뜨는 헤더파일 같은 경우는 돌리기가 불가능 해 보였다. 다음은 컴파일 도중 나타난 주요 문제들이었다.
3. Header-only 파일 같은 경우 pathname으로 다른 헤더 파일을 include하지 않고 헤더 이름만으로 include를 하는 파일이다. 이런 파일들은 직접 컴파일 할때 -I 명령어를 이용하여 파일의 위치를 입력해 줘야 한다. 그 solution 으로 모든 dependency 파일들이 모아 놓은 directory를 Webos 측에서 알려주었다.
4. 하지만 어떤 헤더 파일 같은 경우 github에서 직접 다운로드를 받아 헤더를 include하는 파일도 더러 존재했다. 그 경우 직접 find 명령어를 통해서 찾았어야 했는데 시간도 오래 걸릴 뿐더러 모든 헤더 파일에 대해 전부 찾아보기에는 무리가 있어 보였다.
5. 또 다른 오류의 종류로는 어떤 헤더 파일같은 경우 ifndef로 64비트 와 32비트 운영체제를 판별하는 코드가 존재했다. 이때 64비트로 빌드했을 경우 64비트 헤더 파일만 존재하기 때문에 ABICC는 32비트 헤더 파일이 존재 하지 않는다고 오류가 났었다. 이 경우 ifndef를 명시를 해줬어야 했다.

**9. Future Work**

이후 추가적으로 가능한 변경 사항으로는 다음을 생각해보았습니다.

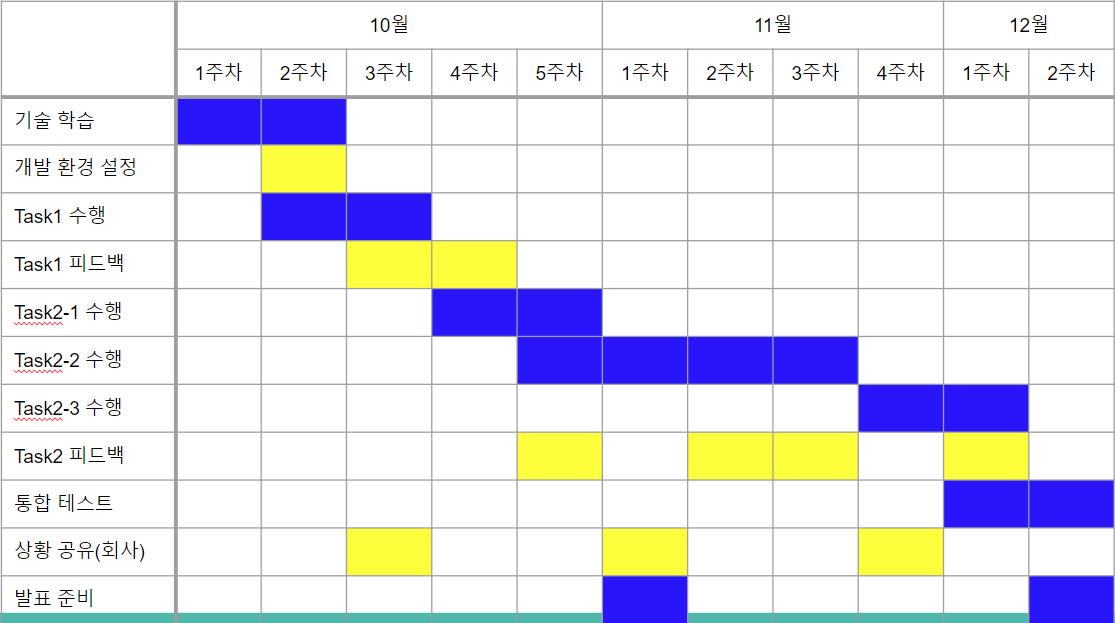
1. WebOS 빌드 시작 조건에 현재 release 탭에서의 업데이트 발생만이 있는데, 다른 변경에서도 대응할 수 있도록 다른 탭이나 commit에 의해서도 빌드를 진행할지 여부를 결정하게끔 수정.

2. 웹 페이지 UI, 관리자 페이지 추가, 필요에 따라 회원제로 진행, JIRA 이슈에 추가적인 내용 변경 등.

**10. Division & Assignment of Work**



**11. Schedule**



# **[Reference]**

1. 김민호, “ LG전자, 독자 플랫폼 webOS 생태계 키운다”, 매일뉴스, 2018/12/07, < <http://www.maeilnews.co.kr/mobile/article.html?no=1340> >

2. Wikipedia, “웹OS”, < <https://ko.wikipedia.org/wiki/%EC%9B%B9OS> >

3. Steven Hilton, “"플랫폼 선택을 위한 필수 지식"··· IoT 마이크로서비스의 6가지 범주”, CIO, 2018/04/20, < <http://www.ciokorea.com/news/37983> >

4. WebOS, “Architecture Overview”, < <https://www.webosose.org/docs/guides/core-topics/architecture/architecture-overview/> >

5. ABI compliance checker, “Detectable Binary Compatibility Problems”, < <https://lvc.github.io/abi-compliance-checker/> >