C++用S3 SDK 读写 OCI Buckets

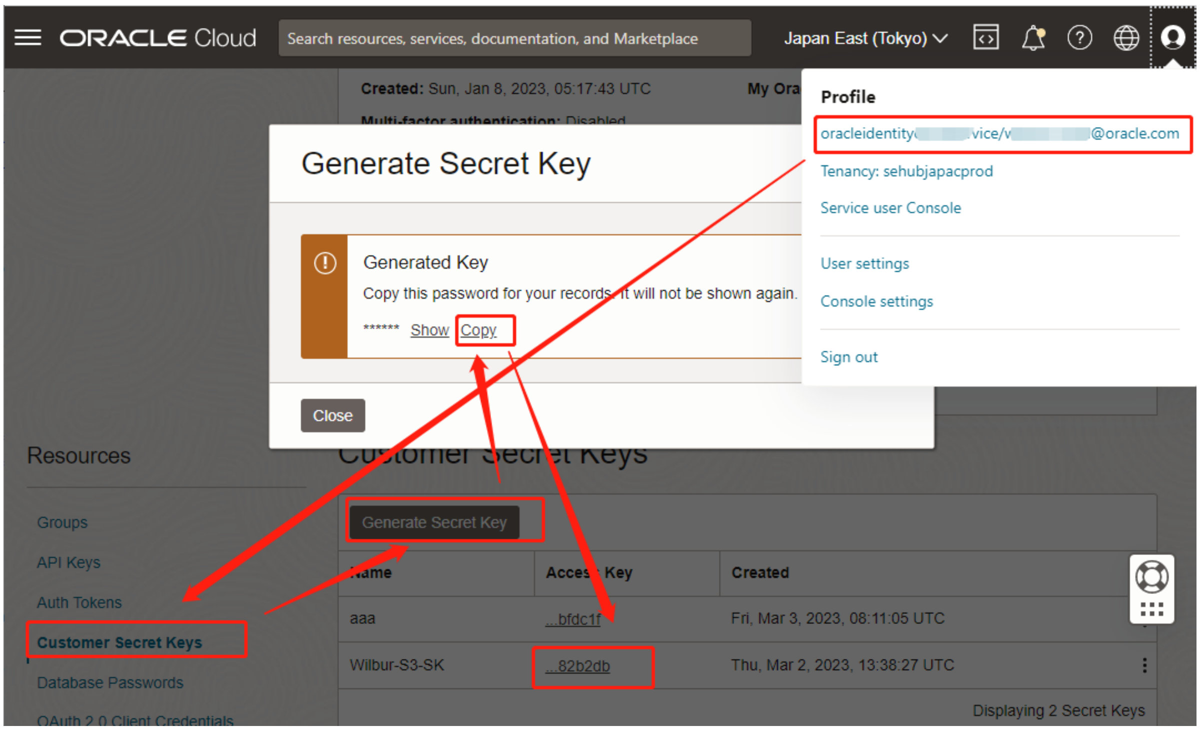
准备工作

Step 1. 准备虚拟机

VM OS: Centos 7

Step 2. 准备S3密钥

准备Customer Key:



Copy this password for your records. It will not be shown again.

******

Name	Access Key	Created
aaa	...bfdc1f	Fri, Mar 3, 2023, 08:11:05 UTC
Wilbur-S3-SK	...82b2db	Thu, Mar 2, 2023, 13:38:27 UTC

把下面2行放到 ~/.bash_profile的末尾：

```
export AWS_ACCESS_KEY_ID=<上图的Access Key>
export AWS_SECRET_ACCESS_KEY=<上图的Secret Key>
```

应用环境变量

```
source ~/.bash_profile
```

安装环境

```
sudo su
setenforce 0
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
systemctl disable firewalld
systemctl stop firewalld

yum install -y  git gcc gcc-c++ make automake libcurl-devel openssl-devel
libuuid-devel pulseaudio-libs-devel
```

```
su opc
mkdir ~/c++/
cd ~/c++/
wget https://cmake.org/files/v3.21/cmake-3.21.0.tar.gz
tar xzvf cmake-3.21.0.tar.gz
cd cmake-3.21.0
./bootstrap && make && sudo make install
```

Step 2. 编译AWS SDK

```
cd ~/c++
git clone --recurse-submodules https://github.com/aws/aws-sdk-cpp
mkdir sdk_build
cd sdk_build
cmake ../aws-sdk-cpp -DCMAKE_BUILD_TYPE=Debug -DCMAKE_PREFIX_PATH=/usr/local/ -
DCMAKE_INSTALL_PREFIX=/usr/local/ -DBUILD_SHARED_LIBS=on -DBUILD_ONLY="s3" -
DENABLE_TESTING=OFF

make && sudo make install
```

验证基本的读写能力

```
mkdir ~/c++/test
cd ~/c++/test
vim test.cpp
```

编写 test.cpp

```
#include <iostream>
#include <fstream>
#include <sys/stat.h>
#include <aws/core/Aws.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/PutObjectRequest.h>
#include <aws/s3/model/GetObjectRequest.h>
using namespace std;

const Aws::String ORACLE_REGION = "ap-tokyo-1";
const Aws::String ORACLE_NAMESPACE = "sehubjapacprod";
const Aws::String ORACLE_BUCKET = "Wilbur-Bucket";

Aws::S3::S3Client getS3Client(){
    Aws::String endpoint = "https://" + ORACLE_NAMESPACE +
".compat.objectstorage." + ORACLE_REGION + ".oraclecloud.com/" + ORACLE_BUCKET +
"/";

    Aws::Client::ClientConfiguration clientConfig;
    clientConfig.verifySSL = false;
    clientConfig.region = ORACLE_REGION;
    clientConfig.endpointOverride = endpoint;

    Aws::S3::S3Client s3_client(clientConfig,
Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never, false);
```

```
    return s3_client;
}

void uploadFile(Aws::S3::S3Client &s3, const Aws::String &keyName, const
Aws::String &sourceFile){
    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(ORACLE_BUCKET);
    request.SetKey(keyName);

    std::shared_ptr<Aws::IOStream> inputData =
        Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
        sourceFile.c_str(),
        std::ios_base::in | std::ios_base::binary);

    if (!*inputData) {
        std::cerr << "Error unable to read file " << sourceFile << std::endl;
        return ;
    }
    request.SetBody(inputData);

    Aws::S3::Model::PutObjectOutcome outcome = s3.PutObject(request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error: PutObject: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Added object '" << sourceFile << "' to bucket '" <<
ORACLE_BUCKET << "'." << std::endl;
    }
}

void downloadFile(Aws::S3::S3Client &s3, const Aws::String &keyName, const
Aws::String &destFile){
    Aws::S3::Model::GetObjectRequest request;
    request.SetBucket(ORACLE_BUCKET);
    request.SetKey(keyName);

    Aws::S3::Model::GetObjectOutcome outcome = s3.GetObject(request);

    if (!outcome.IsSuccess()) {
        const Aws::S3::S3Error &err = outcome.GetError();
        std::cerr << "Error: GetObject: " <<
                  err.GetExceptionName() << ": " << err.GetMessage() <<
std::endl;
    }
    else {
        std::cout << "Successfully retrieved '" << keyName << "' from '" <<
ORACLE_BUCKET << "'." << std::endl;
        std::ofstream outFile;
        outFile.open(destFile);
        outFile << outcome.GetResult().GetBody().rdbuf();
        outFile.close();
    }

    std::cout << "Added object '" << destFile;
```

```
}

int main(){
    Aws::SDKOptions options;
    options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options);
    {
        Aws::S3::S3Client s3 = getS3Client();
        uploadFile(s3, "a5.txt","a1.txt");
        downloadFile(s3, "a5.txt","a5_down1.txt");
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

生成编译文件

```
vi CMakeLists.txt
```

写入

```
cmake_minimum_required(VERSION 3.3)
set(CMAKE_CXX_STANDARD 11)
project(test LANGUAGES CXX)

message(STATUS "CMAKE_PREFIX_PATH: ${CMAKE_PREFIX_PATH}")
set(BUILD_SHARED_LIBS ON CACHE STRING "Link to shared libraries by default.")

#Load required services/packages: This basic example uses S3.
find_package(AWSSDK REQUIRED COMPONENTS s3)
add_executable(${PROJECT_NAME} "test.cpp")

set_compiler_flags(${PROJECT_NAME})
set_compiler_warnings(${PROJECT_NAME})
target_link_libraries(${PROJECT_NAME} ${AWSSDK_LINK_LIBRARIES})
```

编译与测试

```
cmake ./
make
./test
```

验证multipart upload api能力

```
mkdir ~/c++/test2
cd ~/c++/test2
vim test.cpp
```

编写 test.cpp

```
#include <aws/core/Aws.h>
#include <aws/core/utils/UUID.h>
#include <aws/core/utils/json/JsonSerializer.h>
#include <aws/core/auth/AWSCredentialsProvider.h>
#include <aws/core/client/ClientConfiguration.h>
#include <aws/s3/S3Client.h>
#include <aws/s3/model/ListObjectsV2Request.h>
#include <aws/s3/model/PutObjectRequest.h>
#include <aws/s3/model/HeadObjectRequest.h>
#include <aws/s3/model/DeleteObjectRequest.h>
#include <aws/s3/model/CreateMultipartUploadRequest.h>
#include <aws/s3/model/UploadPartRequest.h>
#include <aws/s3/model/CompleteMultipartUploadRequest.h>
#include <aws/s3/model/AbortMultipartUploadRequest.h>
#include <aws/s3/model/ListMultipartUploadsRequest.h>
#include <aws/s3/model/ListPartsRequest.h>
#include <aws/core/utils/memory/stl/AWSString.h>
#include <aws/core/utils/memory/stl/AWSString.h>
#include <aws/core/utils/logging/DefaultLogSystem.h>
#include <aws/core/utils/logging/AWSLogging.h>

#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstring>
#include <curl/curl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

//#define AWS_UPLOAD_PART_SIZE                (5 * 1024 * 1024) // 5MB
#define AWS_UPLOAD_PART_SIZE                (10) // 10 Byte

const Aws::String ORACLE_REGION = "ap-chuncheon-1";
const Aws::String ORACLE_NAMESPACE = "sehubjapacprod";
const Aws::String ORACLE_BUCKET = "Wilbur-Bucket";

Aws::S3::S3Client getS3Client(){
    Aws::String endpoint = "https://" + ORACLE_NAMESPACE +
".compat.objectstorage." + ORACLE_REGION + ".oraclecloud.com/";

    Aws::Client::ClientConfiguration clientConfig;
    clientConfig.verifySSL = false;
    clientConfig.region = ORACLE_REGION;
    clientConfig.endpointOverride = endpoint;

    Aws::S3::S3Client s3_client(clientConfig,
Aws::Client::AWSAuthV4Signer::PayloadSigningPolicy::Never, false);
    return s3_client;
}

void uploadFile(Aws::S3::S3Client &s3, const Aws::String &sourceFile, const
Aws::String &keyName){
    Aws::S3::Model::PutObjectRequest request;
    request.SetBucket(ORACLE_BUCKET);
    request.SetKey(keyName);

    std::shared_ptr<Aws::IOStream> inputData =
        Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
        sourceFile.c_str(),
        std::ios_base::in | std::ios_base::binary);

    if (!*inputData) {
        std::cerr << "Error unable to read file " << sourceFile << std::endl;
        return ;
    }
    request.SetBody(inputData);

    Aws::S3::Model::PutObjectOutcome outcome = s3.PutObject(request);

    if (!outcome.IsSuccess()) {
        std::cerr << "Error: PutObject: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    else {
        std::cout << "Added object '" << sourceFile << "' to bucket '" <<
ORACLE_BUCKET << "'." << std::endl;
    }
}


int aws_s3_multipart_upload(Aws::S3::S3Client &s3_client, const char *local_file,
const char *desc_file) {
    bool new_req_upload = true;
    int partNumber = 0;
    Aws::String uploadId;

    /* 打开文件流 */
    std::shared_ptr<Aws::IOStream> inputData =
        Aws::MakeShared<Aws::FStream>("SampleAllocationTag",
                                      local_file,
                                      std::ios_base::in | std::ios_base::binary);
    if (!*inputData) {
        printf("Read uploaded file failed!\n");
        return -1;
    }

    /* 获取文件大小 */
    inputData->seekg(0, std::ios::end);
    auto fileSize = inputData->tellg();
    inputData->seekg(0, std::ios::beg);
    // 计算文件需要分成多少块上传
    int totalParts = static_cast<int>((static_cast<double>(fileSize) +
        AWS_UPLOAD_PART_SIZE - 1) / AWS_UPLOAD_PART_SIZE);

    // 初始化分块列表
    std::vector<Aws::S3::Model::CompletedPart> completedParts(totalParts);

    /* 获取该文件已上传的块序号 */
    // 构造 ListMultipartUploads 请求
    Aws::S3::Model::ListMultipartUploadsRequest listMultiPartUploadsReq;
    listMultiPartUploadsReq.SetBucket(ORACLE_BUCKET);
    // 发送 ListMultipartUploads 请求
    auto listMultipartUploadsOutcome =
s3_client.ListMultipartUploads(listMultiPartUploadsReq);

    if (listMultipartUploadsOutcome.IsSuccess()) {
        // 获取未完成的分块上传任务列表
        auto list_multipart_uploads_result =
listMultipartUploadsOutcome.GetResult();
        for (const auto& upload : list_multipart_uploads_result.GetUploads()) {
            // 检查上传任务是否匹配
            if (upload.GetKey() == desc_file) {
                new_req_upload = false;
                // 获取上传 ID
                uploadId = upload.GetUploadId();
                // 使用 uploadId 执行后续操作，如获取已上传的块、继续上传等
                Aws::S3::Model::ListPartsRequest listPartsReq;
                listPartsReq.SetBucket(ORACLE_BUCKET);
                listPartsReq.SetKey(desc_file);
                listPartsReq.SetUploadId(uploadId);
                auto listPartsOutcome = s3_client.ListParts(listPartsReq);
                if (listPartsOutcome.IsSuccess()) {
                    auto listPartsRet = listPartsOutcome.GetResult();
                    for (const auto& part : listPartsRet.GetParts()) {
                        // 处理已上传的块信息
                        partNumber = part.GetPartNumber() - 1;
                        inputData->seekg(AWS_UPLOAD_PART_SIZE, std::ios::cur);
                        // 将已完成的分块添加到分块列表
                        Aws::S3::Model::CompletedPart completedPart;
                        completedPart.SetETag(part.GetETag());
                        completedPart.SetPartNumber(partNumber + 1);
                        completedParts[partNumber] = completedPart;
                    }
                    if(listPartsRet.GetParts().size() == 0) {
                        new_req_upload = true;
                        continue;
                    } else {
                        partNumber++;
                    }
                } else {
                    printf("s3_client.ListParts() failed: %s\n",
                        listPartsOutcome.GetError().GetMessage().c_str());
                    return -1;
                }
                break;
            }
        }
    } else {
        printf("ListMultipartUploads failed: %s\n",
            listMultipartUploadsOutcome.GetError().GetMessage().c_str());
        return -1;
    }

    if(new_req_upload) {
        /* 创建分块上传请求 */
        Aws::Map<Aws::String, Aws::String> metadata;
        metadata.emplace("s3-test", "test");
        Aws::S3::Model::CreateMultipartUploadRequest createMultipartUploadReq;
        createMultipartUploadReq.SetBucket(ORACLE_BUCKET);
        createMultipartUploadReq.SetKey(desc_file);
        createMultipartUploadReq.SetMetadata(metadata);
        auto createMultipartUploadOutcome =
            s3_client.CreateMultipartUpload(createMultipartUploadReq);
        if (!createMultipartUploadOutcome.IsSuccess()) {
            printf("Create block to upload failed: %s",
                createMultipartUploadOutcome.GetError().GetMessage().c_str());
            return -1;
        }
        // 获取上传ID和响应的ETag值
        uploadId = createMultipartUploadOutcome.GetResult().GetUploadId();
        printf("Start to be uploaded file.\n");
    } else if(!new_req_upload) {
        printf("Continue uploading file.\n");
    }

    /* 开始分块上传 */
    printf("Upload id = %s\n", uploadId.c_str());
    Aws::S3::Model::UploadPartOutcome uploadPartOutcome;
    for(int p_n = partNumber; p_n < totalParts; ++p_n) {
        // 读取分块数据
        std::vector<char> buffer(AWS_UPLOAD_PART_SIZE);
        inputData->read(buffer.data(), AWS_UPLOAD_PART_SIZE);
        std::streamsize bytesRead = inputData->gcount();
        printf("??????? %.2f MB ??? %d, ? %d ?\n", (float)bytesRead / 1024 /
1024, p_n + 1, totalParts);
        // 上传分块
        Aws::S3::Model::UploadPartRequest uploadPartRequest;
        uploadPartRequest.SetBucket(ORACLE_BUCKET);
        uploadPartRequest.SetKey(desc_file);
        uploadPartRequest.SetPartNumber(p_n + 1);
        uploadPartRequest.SetUploadId(uploadId);
        // 设置分块数据
        std::shared_ptr<Aws::IOStream> partStream =
            Aws::MakeShared<Aws::StringStream>("SampleApp");
        partStream->write(buffer.data(), bytesRead);
        uploadPartRequest.SetBody(partStream);
        uploadPartOutcome = s3_client.UploadPart(uploadPartRequest);
        if (!uploadPartOutcome.IsSuccess()) {
            printf("Upload block %d failed: %s\n", p_n + 1,
                uploadPartOutcome.GetError().GetMessage().c_str());
            return -1;
        }
        // 将已完成的分块添加到分块列表
        Aws::S3::Model::CompletedPart completedPart;
        completedPart.SetETag(uploadPartOutcome.GetResult().GetETag());
        completedPart.SetPartNumber(p_n + 1);
        completedParts[p_n] = completedPart;

        // !!!!!!!!  debug  !!!!!!!!
        //if(p_n == 2) return 0;
    }
    /* 完成分块上传，进行收尾 */
    Aws::S3::Model::CompleteMultipartUploadRequest compMultiPartUploadReq;
    compMultiPartUploadReq.SetBucket(ORACLE_BUCKET);
    compMultiPartUploadReq.SetKey(desc_file);
    compMultiPartUploadReq.SetUploadId(uploadId);
    Aws::S3::Model::CompletedMultipartUpload completedUpload;
    completedUpload.SetParts(completedParts);
    compMultiPartUploadReq.WithMultipartUpload(completedUpload);

    auto completeMultipartUploadOutcome =
s3_client.CompleteMultipartUpload(compMultiPartUploadReq);
    if (completeMultipartUploadOutcome.IsSuccess()) {
        printf("uploaded successfully!\n");
        for(int i = 0; i < completedParts.size(); i++) {
            Aws::S3::Model::CompletedPart part = completedParts[i];
            printf("Block %d ETag: %s\n", part.GetPartNumber(),
part.GetETag().c_str());
        }
    } else {
        printf("Complete block upload failed: %s\n",
            completeMultipartUploadOutcome.GetError().GetMessage().c_str());
        return -1;
    }
    // 关闭文件流
    inputData.reset();

    return 0;
}


int main(){
    Aws::SDKOptions options;
    options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options);
    {
        Aws::S3::S3Client s3 = getS3Client();
        uploadFile(s3 ,"local1.txt", "server1-6.txt");
        aws_s3_multipart_upload(s3,"local2.txt","server2-6.txt");
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

生成编译文件

```
vi CMakeLists.txt
```

写入

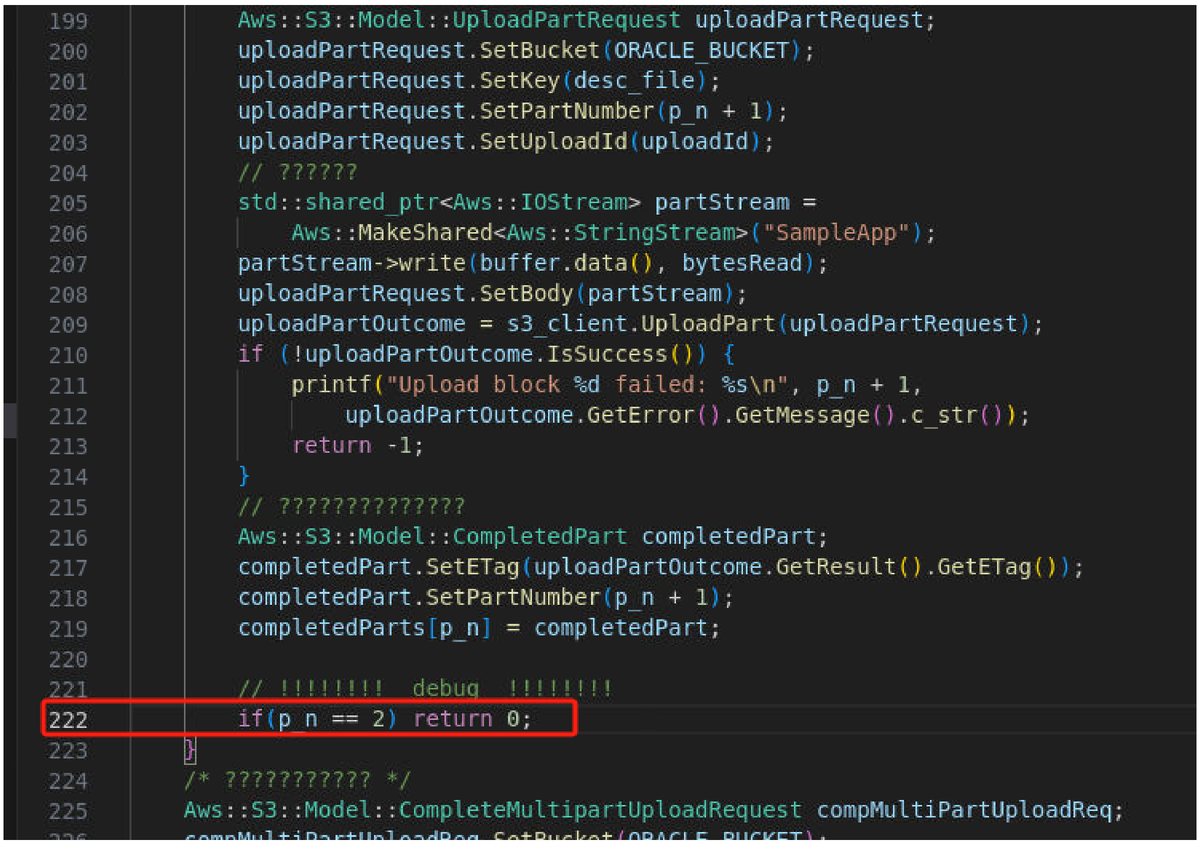
```
cmake_minimum_required(VERSION 3.3)
set(CMAKE_CXX_STANDARD 11)
project(test LANGUAGES CXX)
```

```
message(STATUS "CMAKE_PREFIX_PATH: ${CMAKE_PREFIX_PATH}")
set(BUILD_SHARED_LIBS ON CACHE STRING "Link to shared libraries by default.")

#Load required services/packages: This basic example uses S3.
find_package(AWSSDK REQUIRED COMPONENTS s3)
add_executable(${PROJECT_NAME} "test.cpp")

set_compiler_flags(${PROJECT_NAME})
set_compiler_warnings(${PROJECT_NAME})
target_link_libraries(${PROJECT_NAME} ${AWSSDK_LINK_LIBRARIES})
```

编译与测试

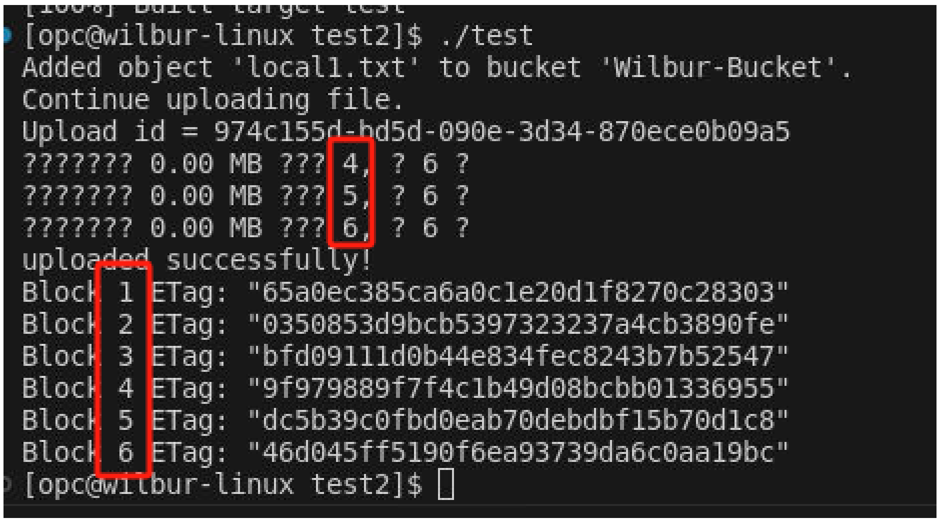
```
cmake ./
make
./test
```

如果要控制传输到一半就断开，注意打开测试开关：



```
        Aws::S3::Model::UploadPartRequest uploadPartRequest;
        uploadPartRequest.SetBucket(ORACLE_BUCKET);
        uploadPartRequest.SetKey(desc_file);
        uploadPartRequest.SetPartNumber(p_n + 1);
        uploadPartRequest.SetUploadId(uploadId);
        // ??????
        std::shared_ptr<Aws::IOStream> partStream =
            Aws::MakeShared<Aws::StringStream>("SampleApp");
        partStream->write(buffer.data(), bytesRead);
        uploadPartRequest.SetBody(partStream);
        uploadPartOutcome = s3_client.UploadPart(uploadPartRequest);
        if (!uploadPartOutcome.IsSuccess()) {
            printf("Upload block %d failed: %s\n", p_n + 1,
                uploadPartOutcome.GetError().GetMessage().c_str());
            return -1;
        }
        // ??????????????
        Aws::S3::Model::CompletedPart completedPart;
        completedPart.SetETag(uploadPartOutcome.GetResult().GetETag());
        completedPart.SetPartNumber(p_n + 1);
        completedParts[p_n] = completedPart;

        // !!!!!!!!  debug  !!!!!!!!
        if(p_n == 2) return 0;
    }
    /* ??????????? */
    Aws::S3::Model::CompleteMultipartUploadRequest compMultiPartUploadReq;
    compMultiPartUploadReq.SetBucket(ORACLE_BUCKET);
```

续传的时候，需要关闭开关，并重新编译且重新执行，就会发现它只续传了剩下的部分：



```
[opc@wilbur-linux test2]$ ./test
Added object 'local1.txt' to bucket 'Wilbur-Bucket'.
Continue uploading file.
Upload id = 974c155d-bd5d-090e-3d34-870ece0b09a5
??????? 0.00 MB ??? 4, ? 6 ?
??????? 0.00 MB ??? 5, ? 6 ?
??????? 0.00 MB ??? 6, ? 6 ?
uploaded successfully!
Block 1 ETag: "65a0ec385ca6a0c1e20d1f8270c28303"
Block 2 ETag: "0350853d9bcb5397323237a4cb3890fe"
Block 3 ETag: "bfd09111d0b44e834fec8243b7b52547"
Block 4 ETag: "9f979889f7f4c1b49d08bcbb01336955"
Block 5 ETag: "dc5b39c0fbd0eab70debdbf15b70d1c8"
Block 6 ETag: "46d045ff5190f6ea93739da6c0aa19bc"
[opc@wilbur-linux test2]$
```