IBM Education Assistance for z/OS V2R1

Item: BSAM type=blocked Support
Element/Component: Language Environment

Material is current as of June 2013

Agenda

- Trademarks
- Presentation Objectives
- Overview
- Usage & Invocation
- Presentation Summary
- Appendix

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.

Presentation Objectives

- Explain the purpose of work on Language Environment BSAM type=blocked support

Overview

- Problem Statement / Need Addressed
  - The C runtime library I/O interface does not allow for high performance processing of sequential data sets where the desire is to copy/transfer a data set from one location to another.
  - Although there is support for opening a fixed or variable record format as undefined in read mode, there is no support for doing the same in write mode, which would dramatically speed up this kind of processing.

Overview

- Solution
  - Support reading, writing, and repositioning of sequential data sets by blocks, rather than by bytes or records
- Benefit / Value
  - Improve the data set operation performance of XL C/C++ runtime library when there is no need to manipulate data within the blocks.

Usage & Invocation

- Using this support, you can:
  - Use XL C/C++ runtime library routines to process sequential data sets using blocked I/O.
  - Read from, write to, and reposition (seek) within data sets by blocks.
- Typical scenario: copy a sequential data set from one place to another:
  - Open source/destination data sets with “type=blocked”
  - Read one block from the source and write it to the destination
  - Loop step 2 until EOF
  - Close files

Usage & Invocation

- The support is invoked by calling the fopen()/freopen() functions on a sequential data set with keyword parameter “type=blocked” specified.

```
FILE *stream;

stream = fopen("//USR01.TMP",

         "rb+, lrecl=80,blksize=240, recfm=VB, type=blocked");
```

- Once the data set is opened, other I/O functions can be used to process the stream.
  - fread()/fwrite()
  - rewind()/ftell()/fseek()/ftello()/fseeko()/fgetpos()/fsetpos()
  - fflush()/fldata()/fclose()

Usage & Invocation

- Only binary open mode on BSAM I/O is supported for blocked I/O
- Byte I/O functions below are not supported for blocked I/O
  - fgetc()/fgets()
  - fputc()/fputs()
  - fprintf()/printf()/sprintf()
  - fscanf()/scanf()/sscanf()
  - fwide()
  - getc()/getchar()/gets()
  - getc_unlocked()/getchar_unlocked()/putc_unlocked()/putchar_unlocked()
  - putc()/putchar()/puts()/putwchar()
  - ungetc()/ungetwc()
  - vfprintf()/vprintf()

Usage & Invocation

- Buffering
  - For blocked I/O files, buffering is always meaningless.
    - A block is written out as soon as fwrite() completes.
    - The function fflush() has no effect for files opened with “type=blocked”.

Usage & Invocation

- Reading from files
  - For files opened in blocked format, fread() is the only interface that supports reading.
  - Each time you call fread() for a blocked I/O file, fread() reads one block.
    - If calling fread() with a request for less than a complete block, the requested bytes are copied to your buffer, and the file position is set to the start of the next block.
    - If the request is for more bytes than are in the block, one block is read and the position is set to the start of the next block. z/OS XL C/C++ does not strip any blank characters or interpret any data.

Usage & Invocation

- Reading from files
  - fread() returns the number of items read successfully.
    - If passing a **size** argument equal to 1 and a **count** argument equal to the maximum expected length of the block, fread() returns the length, in bytes, of the block read.
    - If passing a **size** argument equal to the maximum expected length of the block, and a **count** argument equal to 1, fread() returns either 0 or 1, indicating whether a block of length size read. If a block is read successfully but is less than size bytes long, fread() returns 0.
  - A failed read operation may lead to undefined behavior until you reposition successfully.

Usage & Invocation

- Writing to files
  - fwrite() is the only interface allowed for writing to a file opened for blocked I/O. Only one block is written at a time.
  - Writing more than BLKSIZE bytes
    - The data will be truncated.
  - Writing less than BLKSIZE bytes
    - If to create a new block, a short block will be created.
    - If to update an existing block, only requested part of the block will be updated.
  - z/OS XL C/C++ will not check the data provided by the user.
  - At the completion of an fwrite(), the file position is at the start of the next block, and the block is flushed out to the system.

Usage & Invocation

- Writing to files - special behaviors
  - Because all fixed-format records must be full, any block you write must be multiple of a record, else z/OS XL C/C++ will fail the write request.
  - When writing or appending to a FBS short block at the end of file, z/OS XL C/C++ will use the request buffer to replace the previous block, and that may extend or shrink the short block.
  - When updating a FBS short block at the end of file, it could be updated to a full block or a longer short block from start of the short block.
  - It is user's responsibility to make sure there is no short block in the middle of a FBS data set.
  - It is users' responsibility to make sure BDWs, RDWs and SDWs in a block of variable record format file are correct.

Usage & Invocation

- Repositioning within files
  - ftell() returns relative block numbers.
  - The behavior of fseek() and ftell() is similar to that when you use relative byte offsets for binary files, except that the unit is a block rather than a byte.
    - For example,

      `fseek(fp,-2,SEEK_CUR);`

      seeks backward two blocks from the current position.
  - You cannot seek past the end or before the beginning of a file.
  - The first block of a file is relative block 0.
  - For AMODE 31 C/C++ applications repositioning within a large format sequential data set that needs to access positions beyond 2GB-1block must use the large file version of fseeko() and ftello().

Presentation Summary

- XL C/C++ runtime library routines support BSAM (seek) processing of data sets under blocked I/O.

Appendix

- Publications:
  - z/OS XL C/C++ Runtime Library Reference (SA22-7821)
  - z/OS XL C/C++ Programming Guide (SC09-4765)
  - z/OS Language Environment Run-Time Messages (SC22-7566)