```
/*
  bsdiff.c -- Binary patch generator.

  Copyright 2003 Colin Percival

  For the terms under which this work may be distributed, please see
  the adjoining file "LICENSE".

  ChangeLog:
  2005-05-05 - Use the modified header struct from bspatch.h; use 32-bit
               values throughout.
                --Benjamin Smedberg <benjamin@smedbergs.us>
  2005-05-18 - Use the same CRC algorithm as bzip2, and leverage the CRC table
               provided by libbz2.
                --Darin Fisher <darin@meer.net>
  2007-11-14 - Changed to use Crc from Lzma library instead of Bzip library
                --Rahul Kuchhal
*/

#include "mbspatch.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <stdarg.h>
#ifdef _WIN32
#include <io.h>
#include <winsock2.h>
#else
#include <unistd.h>
#include <arpa/inet.h>
#define _O_BINARY 0
#endif

#undef MIN
#define MIN(x,y) (((x)<(y)) ? (x) : (y))

static void
reporterr(int e, const char *fmt, ...)
{
        if (fmt) {
                va_list args;
                va_start(args, fmt);
                vfprintf(stderr, fmt, args);
                va_end(args);
        }

        exit(e);
}

static void
split(int *I,int *V,int start,int len,int h)
{
        int i,j,k,x,tmp,jj,kk;
```

```
	if(len<16) {
		for(k=start;k<start+len;k+=j) {
			j=1;x=V[l[k]+h];
			for(i=1;k+i<start+len;i++) {
				if(V[l[k+i]+h]<x) {
					x=V[l[k+i]+h];
					j=0;
				};
				if(V[l[k+i]+h]==x) {
					tmp=l[k+j];l[k+j]=l[k+i];l[k+i]=tmp;
					j++;
				};
			};
			for(i=0;i<j;i++) V[l[k+i]]=k+j-1;
			if(j==1) l[k]=-1;
		};
		return;
	};

	x=V[l[start+len/2]+h];
	jj=0;kk=0;
	for(i=start;i<start+len;i++) {
		if(V[l[i]+h]<x) jj++;
		if(V[l[i]+h]==x) kk++;
	};
	jj+=start;kk+=jj;

	i=start;j=0;k=0;
	while(i<jj) {
		if(V[l[i]+h]<x) {
			i++;
		} else if(V[l[i]+h]==x) {
			tmp=l[i];l[i]=l[jj+j];l[jj+j]=tmp;
			j++;
		} else {
			tmp=l[i];l[i]=l[kk+k];l[kk+k]=tmp;
			k++;
		};
	};

	while(jj+j<kk) {
		if(V[l[jj+j]+h]==x) {
			j++;
		} else {
			tmp=l[jj+j];l[jj+j]=l[kk+k];l[kk+k]=tmp;
			k++;
		};
	};

	if(jj>start) split(l,V,start,jj-start,h);

	for(i=0;i<kk-jj;i++) V[l[jj+i]]=kk-1;
	if(jj==kk-1) l[jj]=-1;

	if(start+len>kk) split(l,V,kk,start+len-kk,h);
}
```

```
static void
qsufsort(int *I,int *V,unsigned char *old,int oldsize)
{
	int buckets[256];
	int i,h,len;

	for(i=0;i<256;i++) buckets[i]=0;
	for(i=0;i<oldsize;i++) buckets[old[i]]++;
	for(i=1;i<256;i++) buckets[i]+=buckets[i-1];
	for(i=255;i>0;i--) buckets[i]=buckets[i-1];
	buckets[0]=0;

	for(i=0;i<oldsize;i++) I[++buckets[old[i]]]=i;
	I[0]=oldsize;
	for(i=0;i<oldsize;i++) V[i]=buckets[old[i]];
	V[oldsize]=0;
	for(i=1;i<256;i++) if(buckets[i]==buckets[i-1]+1) I[buckets[i]]=-1;
	I[0]=-1;

	for(h=1;I[0]!=-(oldsize+1);h+=h) {
		len=0;
		for(i=0;i<oldsize+1;) {
			if(I[i]<0) {
				len-=I[i];
				i-=I[i];
			} else {
				if(len) I[i-len]=-len;
				len=V[I[i]]+1-i;
				split(I,V,i,len,h);
				i+=len;
				len=0;
			};
		};
		if(len) I[i-len]=-len;
	};

	for(i=0;i<oldsize+1;i++) I[V[i]]=i;
}

static int
matchlen(unsigned char *old,int oldsize,unsigned char *newbuf,int newsize)
{
	int i;

	for(i=0;(i<oldsize)&&(i<newsize);i++)
		if(old[i]!=newbuf[i]) break;

	return i;
}

static int
search(int *I,unsigned char *old,int oldsize,
    unsigned char *newbuf,int newsize,int st,int en,int *pos)
{
	int x,y;
```

```
	if(en-st<2) {
		x=matchlen(old+I[st],oldsize-I[st],newbuf,newsize);
		y=matchlen(old+I[en],oldsize-I[en],newbuf,newsize);

		if(x>y) {
			*pos=I[st];
			return x;
		} else {
			*pos=I[en];
			return y;
		}
	};

	x=st+(en-st)/2;
	if(memcmp(old+I[x],newbuf,MIN(oldsize-I[x],newsize))<0) {
		return search(I,old,oldsize,newbuf,newsize,x,en,pos);
	} else {
		return search(I,old,oldsize,newbuf,newsize,st,x,pos);
	};
}

int main(int argc,char *argv[])
{
	int fd;
	unsigned char *old,*newbuf;
	int oldsize,newsize;
	int *I,*V;

	int scan,pos,len;
	int lastscan,lastpos,lastoffset;
	int oldscore,scsc;

	int s,Sf,lenf,Sb,lenb;
	int overlap,Ss,lens;
	int i;

	int dblen,eblen;
	unsigned char *db,*eb;

	unsigned int scrc;

	MBSPatchHeader header = {
		{'M','B','D','I','F','F','1','0'},
		0, 0, 0, 0, 0, 0
	};

	int numtriples;

	if(argc!=4)
		reporterr(1,"usage: %s <oldfile> <newfile> <patchfile>\n",argv[0]);

	/* Allocate oldsize+1 bytes instead of oldsize bytes to ensure
		that we never try to malloc(0) and get a NULL pointer */
	if(((fd=open(argv[1],O_RDONLY|_O_BINARY,0))<0) ||
		((oldsize=lseek(fd,0,SEEK_END))==-1) ||
```

```
        ((old=(unsigned char*) malloc(oldsize+1))==NULL) ||
        (lseek(fd,0,SEEK_SET)!=0) ||
        (read(fd,old,oldsize)!=oldsize) ||
        (close(fd)==-1))
        reporterr(1,"%s\n",argv[1]);

scrc = CalculateCrc(old, oldsize);

if(((I=(int*) malloc((oldsize+1)*sizeof(int)))==NULL) ||
        ((V=(int*) malloc((oldsize+1)*sizeof(int)))==NULL))
        reporterr(1,NULL);

qsufsort(I,V,old,oldsize);

free(V);

/* Allocate newsize+1 bytes instead of newsize bytes to ensure
        that we never try to malloc(0) and get a NULL pointer */
if(((fd=open(argv[2],O_RDONLY|_O_BINARY,0))<0) ||
        ((newsize=lseek(fd,0,SEEK_END))==-1) ||
        ((newbuf=(unsigned char*) malloc(newsize+1))==NULL) ||
        (lseek(fd,0,SEEK_SET)!=0) ||
        (read(fd,newbuf,newsize)!=newsize) ||
        (close(fd)==-1)) reporterr(1,"%s\n",argv[2]);

if(((db=(unsigned char*) malloc(newsize+1))==NULL) ||
        ((eb=(unsigned char*) malloc(newsize+1))==NULL))
        reporterr(1,NULL);

dblen=0;
eblen=0;

if((fd=open(argv[3],O_CREAT|O_TRUNC|O_WRONLY|_O_BINARY,0666))<0)
        reporterr(1,"%s\n",argv[3]);

/* start writing here */

/* We don't know the lengths yet, so we will write the header again
         at the end */

if(write(fd,&header,sizeof(MBSPatchHeader))!=sizeof(MBSPatchHeader))
        reporterr(1,"%s\n",argv[3]);

scan=0;len=0;
lastscan=0;lastpos=0;lastoffset=0;
numtriples = 0;
while(scan<newsize) {
        oldscore=0;

        for(scsc=scan+=len;scan<newsize;scan++) {
                len=search(I,old,oldsize,newbuf+scan,newsize-scan,
                                0,oldsize,&pos);

                for(;scsc<scan+len;scsc++)
                if((scsc+lastoffset<oldsize) &&
                        (old[scsc+lastoffset] == newbuf[scsc]))
```

```
                oldscore++;

        if(((len==oldscore) && (len!=0)) ||
                (len>oldscore+8)) break;

        if((scan+lastoffset<oldsize) &&
                (old[scan+lastoffset] == newbuf[scan]))
                oldscore--;
};

if((len!=oldscore) || (scan==newsize)) {
        MBSPatchTriple triple;

        s=0;Sf=0;lenf=0;
        for(i=0;(lastscan+i<scan)&&(lastpos+i<oldsize);) {
                if(old[lastpos+i]==newbuf[lastscan+i]) s++;
                i++;
                if(s*2-i>Sf*2-lenf) { Sf=s; lenf=i; };
        };

        lenb=0;
        if(scan<newsize) {
                s=0;Sb=0;
                for(i=1;(scan>=lastscan+i)&&(pos>=i);i++) {
                        if(old[pos-i]==newbuf[scan-i]) s++;
                        if(s*2-i>Sb*2-lenb) { Sb=s; lenb=i; };
                };
        };

        if(lastscan+lenf>scan-lenb) {
                overlap=(lastscan+lenf)-(scan-lenb);
                s=0;Ss=0;lens=0;
                for(i=0;i<overlap;i++) {
                        if(newbuf[lastscan+lenf-overlap+i]==
                          old[lastpos+lenf-overlap+i]) s++;
                        if(newbuf[scan-lenb+i]==
                          old[pos-lenb+i]) s--;
                        if(s>Ss) { Ss=s; lens=i+1; };
                };

                lenf+=lens-overlap;
                lenb-=lens;
        };

        for(i=0;i<lenf;i++)
                db[dblen+i]=newbuf[lastscan+i]-old[lastpos+i];
        for(i=0;i<(scan-lenb)-(lastscan+lenf);i++)
                eb[eblen+i]=newbuf[lastscan+lenf+i];

        dblen+=lenf;
        eblen+=(scan-lenb)-(lastscan+lenf);

        triple.x = htonl(lenf);
        triple.y = htonl((scan-lenb)-(lastscan+lenf));
        triple.z = htonl((pos-lenb)-(lastpos+lenf));
        if (write(fd,&triple,sizeof(triple)) != sizeof(triple))
```

```
                                reporterr(1,NULL);

#ifdef DEBUG_bsmedberg
                        printf("Writing a block:\n"
                             "  X: %u\n"
                             "  Y: %u\n"
                             "  Z: %i\n",
                             (int) lenf,
                             (int) ((scan-lenb)-(lastscan+lenf)),
                             (int) ((pos-lenb)-(lastpos+lenf)));
#endif

                        ++numtriples;

                        lastscan=scan-lenb;
                        lastpos=pos-lenb;
                        lastoffset=pos-scan;
                };
        };

        if(write(fd,db,dblen)!=dblen)
                reporterr(1,NULL);

        if(write(fd,eb,eblen)!=eblen)
                reporterr(1,NULL);

        header.slen      = htonl(oldsize);
        header.scrc32    = htonl(scrc);
        header.dlen      = htonl(newsize);
        header.cblen     = htonl(numtriples * sizeof(MBSPatchTriple));
        header.difflen   = htonl(dblen);
        header.extralen = htonl(eblen);

        if (lseek(fd,0,SEEK_SET) == -1 ||
           write(fd,&header,sizeof(header)) != sizeof(header) ||
           close(fd) == -1)
                reporterr(1,NULL);

        free(db);
        free(eb);
        free(l);
        free(old);
        free(newbuf);

        return 0;
}
```